UNIVERSIDAD CARLOS II DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA TELEMÁTICA

TRABAJO FIN DE GRADO

Desarrollo de una aplicación web para el telecontrol de un humedal inteligente

Autor: Rodrigo Argüello Flores

Director en empresa: Jaime Mancebo Galán

Tutor: Jesús Arias Fisteus

Septiembre 2014

Título: Desarrollo de una aplicación web para el telecontrol de un humedal inteligente

Autor: Rodrigo Argüello Flores

Director en empresa: Jaime Mancebo Galán

Tutor: Jesús Arias Fisteus

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día …… en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, el tribunal:

Presidente:

Secretario:

Vocal:

Acuerdan otorgarle la calificación de:

Calificación:

SEPTIEMBRE 2014

Agradecimientos

Me gustaría aprovechar esta sección para agradecer a mi familia y amigos por estar conmigo en todo momento y ayudarme en los momentos más difíciles.

No ha sido un camino fácil llegar hasta aquí pero no lo hubiera conseguido sin vosotros.

A Jesús, mi tutor, uno de los profesores con los que más he aprendido durante mi estancia en la universidad.

A mis padres, que desde el momento en que nací lo han dado todo por mí y sin ellos no sería la persona que soy ahora.

A mi hermanos Moncho, Javier y Jorge, que me han ayudado durante toda mi vida tanto en aspectos académicos como personales.

A Miriam, la persona más especial de mi vida a la cual tengo muchísimo que agradecer y me hace crecer como persona día a día.

A Nister y a Eks, ellos saben quien son y sabe que a pesar de todo son unos grandes y buenos amigos.

Gracias a todos por haberme apoyado y ayudado a llegar hasta aquí.

Resumen

Las tecnologías de la información y de la comunicación (TIC) están presentes en nuestra vida cotidiana y cada vez más, debido a que las conexiones a Internet son cada vez de mayor calidad y más accesibles para todo el mundo. Esto está provocando nuevos avances tecnológicos en nuestra sociedad.

Una de las herramientas más populares en Internet son las aplicaciones web, debido a que todos los usuarios que disponen de conexión a Internet poseen navegadores web instalados en sus equipos. Esto supone que las aplicaciones web son independientes del sistema operativo y no es necesario instalar software ni actualizaciones para el mantenimiento de las mismas.

Otra de las tendencias de los últimos años son las Redes de Sensores Inalámbricos, que están revolucionando el mundo de la tecnología y de Internet. Gracias a esta tecnología, se están consiguiendo avances en terrenos como son la domótica, monitorización de salud o la monitorización ambiental. Este último campo es al que pertenece el TFG descrito en este documento.

En el TFG desarrollado se ha diseñado e implementado una aplicación web capaz de tele controlar un humedal inteligente situado en una población rural, comunicándose con una Red de Sensores Inalámbrica y un sistema de control situados en el humedal.

El objetivo tras esto es el de facilitar las tareas de mantenimiento del mismo, así como de tele controlar el sistema de control asociado a la depuración del mismo, para principalmente conseguir ahorros de coste y mejorar la viabilidad de la implantación y el despliegue masificado de los mismos.

Abstract

Information and Communication Technology (ICT) is part of our daily life and is increasing more and more as Internet connections have more quality and are more accessible to anyone in the world. This entails new technological advances in our society.

One of the most popular tools on the Internet is web applications, as everyone has a web browser already installed in their computers. That involves that web applications are independent from the operative system and it's not necessary to install neither software nor updates for maintenance.

Another important trend in the recent years is Wireless Sensorial Networks, which are overturning technology and the Internet itself. Thanks to this technology, advances in areas such as home automation, health and ambient monitoring are taking place. This last field is where the TFG described here belongs to.

In the developed TFG has been designed and implemented an application able to remote control an intelligent wetland located in a rural population, that communicates with a Wireless Sensorial Network and a control system both located in the wetland.

The whole goal is to ease maintenance tasks of the wetland itself, as well as remote control the control system linked to the depuration of the wetland, mainly for saving costs and improve the implantation viability and the massive deploy of them.

Extended abstract

In this additional chapter, the most important concepts from this document will be exposed. This summarized chapter will resume concepts from the chapters 2 to 7 from the main Spanish document. Introduction and conclusion chapters will not be included in this extended abstract because these chapters are already in English inside the main document.

State of art

The developed task in this Final Project is a web application, but it is also related to some other technological fields because of its connection with the rest of the system, such as Wireless Sensor Networks and some of the latest research jobs in the field of the waste water treatment by biological processes.

Waste water treatment

Waste water treatment state of art, in the field that concerns this project, is the waste water depuration by biological processes, where the constructed wetlands are situated.

Constructed wetlands are artificial wetlands where depurating processes happen in a natural way. The latest researches in this field have brought the incorporation of some additional technologies to accelerate this natural depurating process. This is called bioelectrogenesis whose official website with further information is in [26]. It also uses some other technologies called Microbial Fuel Cells (MFC), whose official website is in [27].

Wireless Sensor Network

Wireless Sensor Network (WSN) is one of the most used technologies nowadays. It consists on a distributed network of sensor nodes, capable to get physical or chemical measures from its environment, process this information with a microcontroller and send it to a central node via wireless communication technologies in order to process this information. The objectives behind this are usually monitoring the environment in order to act in it. A special wireless communication protocol designed for WSN is the IEEE 802.15.4, whose main characteristic for this purpose is its energy efficiency.

Web development state of art

There exist several web development coding languages and frameworks. The most used ones in the web backend (server-side) are PHP, Java, Python, Ruby, ASP.NET, etc. and the most used one in the front end (client-side) is JavaScript.

During this state of art, most important advantages of Python programming language for web development are discussed.

Besides, the Django framework for Python is one of the most used frameworks for web development. It has a lot of powerful tools which are deeply analyzed during this document. It provides the programmer a lot of shortcuts functions for the most repetitive tasks during web development. It is also an interesting choice because of the standard security it offers, protecting websites from some hacker attacks such as SQL Injection or Cross-site scripting.

Other analyzed technology is Node.js, a powerful server-side framework. It is written in the JavaScript language. Its architecture allows creating server programs capable to serve a huge amount of requests at the same time.

It has a lot of external modules to extend the functionality. The most interesting one is socket.io, which allows writing real-time applications. This module implements the same functionality as the Websockets technology, but optimizing it. Applications exchanging messages and being showed to the user without the need to refresh web navigator can be created with these two technologies.

The last discussed technologies are related to the front end web development. These technologies are JavaScript and the latest open standard to transmit data objects

inside web applications. It is substituting XML because of its light and more readable format.

Requirements

A general recapitulation of the most important requirements the system has to satisfy is the following:

- The WSN has to sample a set of previously defined variables every 30 seconds, and also will be able to change some parameters from the wetland control system. This control system will be also available via Internet (web application).
- The remote control system will have the following functionalities: monitoring view, control box view and charts view.
- In addition, the remote control system will be private. All the functionalities from the web application will be only available for registered users.
- Only the administrator will be able to register new users. The administration of the website will be done all through a Graphic Interface. Finally, different roles on the users from the website must be implemented.

Architecture and design

In this chapter, global system, WSN and remote control system architectures are described. Only the remote control architecture was designed in this project. The elements conforming it are the following:

- **Proxy server (*Broker*):** it is proxy server between the web application and the central node from the WSN. The main functionalities performed by this element are listening frames from the WSN to its after process and the redirection of these frames to the Node.js server.
- **Node.js server:** this server performs the real-time viewing functionality of the web application. The Node.js client is inside the HTML monitoring application template.
- **Web application (Django):** the main functionality will be here. It performs the database queries, URLs processing and database writing through admin site and control box sub application. This element is divided into 3 sub applications:
  - **Monitoring** sub application.
  - **Control box** sub application.
  - **Charts** sub application.

- **Web server:** it's in charge to link clients to the requested content from the page.
- **Client:** this is a web browser, capable to make HTTP requests and render the received content. Behind the web browser, the client will be the wetland operator in charge of the depurating control system from the wetland.

Implementation

The main stages of the implementation have been the following:

- Implementation of the web-site broker: TCP mechanisms have been analyzed in order to receive all the data in the proper way, despite of connection errors.
- Implementation of the Node.js client/server: this task has been one of the most important ones. It provides the application the functionality of viewing real-time data. JSON data structures have been created to exchange messages through socket.io and the listener functions have been designed and coded.
- Implementation of the Django application: all the sub-application view functions, HTML templates and models have been coded. Also, settings have been configured and all the details of a Django project.

Testing

Main performed tests to check the communication with the WSN have been made with coded Python scripts and Wireshark.

Whole web application functionality and security has been tested with successful result

Índice de contenidos

Tabla de ilustraciones

Índice de tablas

Chapter 1

Introduction

1.1 MOTIVATION

At national level, in sanitation and water treatment, is pending the challenge in small populations, one area in which huge deficiencies are detected, being estimated between 3 and 4 million of people that still don't receive adequate water treatment in populations under 2000 people. Although this number only represents a few percent compared national level, the number affected estimated of areas is greater than 6000, most of them less than 500 people [20].

Also there are many problems in water treatment in rural areas. These problems are the following [19]:

- Limited technical and economic resources.
- Deficient infrastructure.
- Limited human resources for water treatment plants maintenance.
- As consequence of previously exposed, there is a bad water treatment quality.

The Smart Wetland project (belonging to the INNPACTO 2012 subprogram, from Economy and Competitive Ministry, MINECO, that proposes perform R&D projects oriented to exploitable products based on demand) proposes one feasible solution for the water treatment in rural areas. This project proposes one economic feasible solution, thanks to technologies are going to be explained in chapter 2 of this document.

The developed task for this Final Project, which is exposed in this document, is a subtask of the Smart Wetland project. Therefore, the motivations of this subtask inside this project to improve the quality and viability of these kind of systems are the following:

- One of the main disadvantages of this kind of technologies is, once built, workers cannot act on them to adapt to water flow changes.
- Also, the existing budget limitations in small populations make impractical the hiring of required human resources in a traditional waste water plant.

The aim of this Final Project is to create a web application capable of allowing to remote control the wetland depurating system. This tool will let the workers do the maintenance, monitoring and remote control of the depurating wetland system, allowing the wetland workers not to move to the location of this one to perform this operations. As a direct consequence of this, a cost reduction will happen and the viability of the deployment of these systems will improve.

1.2 OBJECTIVES

The main proposal of this project is to develop a web application that allows to remote control a certain wetland, which is equipped with a direct control system, capable to make physical changes on the wetland, and a sensors network, in charge to obtain water measures and information related to the quality of the depurating process.

More specifically, the expected goals to achieve regarding the functionality of the developed system are:

- The application must display the wetland data in real time. The data will come from the sensors network.
- It also has to allow making changes on some of the wetland control system parameters.
- As an additional functionality, it has to allow viewing charts of the obtained data. This charts will be used by the wetland workers to detect some anomalies in the depurating process.
- Finally, the application must be reliable and secure. It has to permit to control, to a certain admin, the permissions of the registered users.

1.3 ORGANIZATION OF THIS DOCUMENT

This document is divided into some chapters in the following way:

Chapter 1: Introduction

In this chapter, an introduction to the motivations and the context of the project is explained. Also, the defined objectives of the task are exposed. Finally, the content of every chapter of the document is explained.

Chapter 2: State of art

In this chapter, some of the background technologies are discussed. Web development technologies are mainly discussed, but other present technologies in the bigger project are discussed, such as Wireless Sensors Networks and some innovating waste water treatment technologies. Furthermore, precedent R&D projects to this task are exposed.

Chapter 3: Requirements

Requirements of the whole system are exposed in this chapter. The chapter is divided into the requirements of each part of the whole system.

Chapter 4: Architecture

In this chapter, the designed remote control architecture and the system global architecture are exposed. In addition, previously designed (by third parts) architectures involved in the system are displayed.

Chapter 5: Design

Remote control system design is defined in this chapter. Same as previous chapters, the design is exposed separately into the different elements of the system. Also, some design choices are discussed.

Chapter 6: Implementation

The development and implementation process is exposed in this chapter. Some code fragments are showed in order to make an easier understanding.

Chapter 7: Tests

In this chapter, the remote control system test battery is showed. It is divided into networking and messaging testing, functionality testing and, finally, security testing.

Chapter 8: Conclusions and future work

Reached conclusions during the development of the project are exposed in this chapter. In addition, possible future works are discussed.

Capítulo 2

Estado del arte

2.1 TRATAMIENTO DE AGUAS RESIDUALES

El tratamiento de las aguas residuales consiste en una serie de procesos físicos, químicos y biológicos que tienen como objetivo eliminar los contaminantes presentes en el agua procedente de uso humano.

Este tratamiento sigue varias etapas una vez que las aguas llegan a la planta de tratamiento desde las redes de saneamiento. Las etapas principales son:

- Tratamiento primario (asentamiento de sólidos presentes en las aguas).
- Tratamiento secundario (tratamiento biológico de la materia orgánica disuelta en el agua residual. Esta etapa puede ser sustituida por humedales artificiales, que es el punto en el que se desarrolla el proyecto).
- Tratamiento terciario (pasos adicionales como microfiltración, desinfección, etc.).

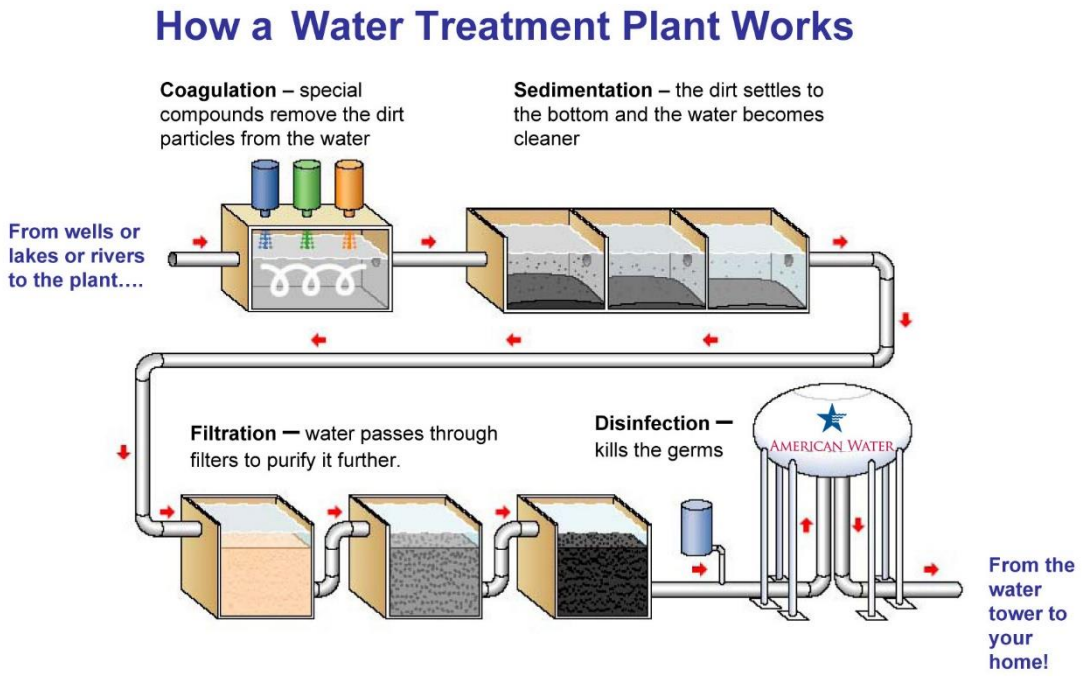

Ilustración 1: Funcionamiento de una planta de tratamiento de aguas residuales. Fuente: http://www.amwater.com/

El proceso natural de la limpieza del agua se consigue gracias a una bacteria que se alimenta de los desechos que contienen las aguas residuales. Gracias a esta bacteria, se consiguen los sistemas de tratamiento de aguas residuales por medios biotecnológicos, en los que, mediante diferentes métodos, se consigue acelerar este proceso natural.

2.1.1 HUMEDALES ARTIFICIALES

Los humedales artificiales son zonas construidas por el hombre en las que se intenta reproducir, de manera controlada, todos los mecanismos de depuración del agua que ocurren de manera natural en los humedales naturales mediante diferentes procesos físicos, biológicos y químicos.

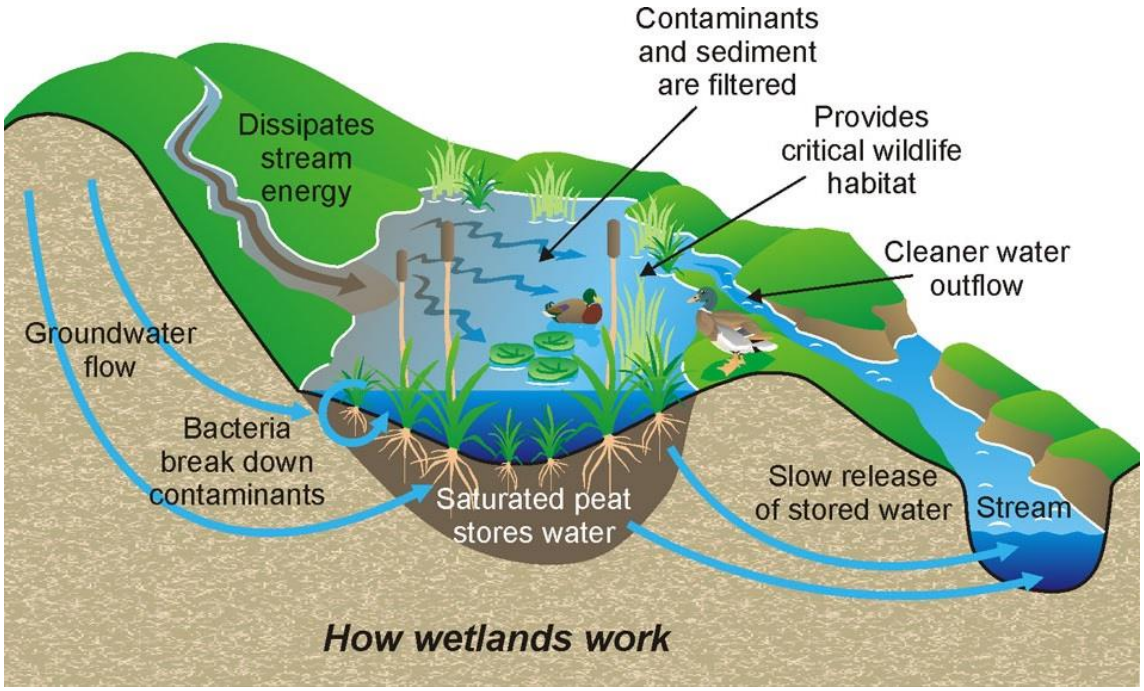

Ilustración 2: Cómo funciona un humedal. Fuente: http://www.taylornoakes.com/

Los procesos de depuración del agua que se dan en los humedales son los siguientes:

- Eliminación de sedimentos gracias a la filtración que se da entre el sustrato y la vegetación del sistema.
- Eliminación de materia orgánica gracias a la acción de los microorganismos (principalmente bacterias).
- Eliminación de nitrógeno gracias a la acción directa de las plantas y de los microorganismos.
- Eliminación de fósforo gracias a la absorción del sustrato.
- Eliminación de patógenos gracias a todos los elementos involucrados en el sistema (sustratos, vegetación, microorganismos).

Durante décadas, el uso que se le ha dado a este tipo de sistemas es para el tratamiento de aguas residuales urbanas e industriales.

En la actualidad, se está investigando la implantación de este tipo de sistemas, con tecnología añadida, para su uso en la depuración de aguas residuales de pequeñas poblaciones, debido a que es una alternativa aceptable a las plantas de depuración convencionales, tanto económica como ecológicamente, ya que son sistemas de fácil construcción, bajo costo y mantenimiento reducido.

2.1.2 DEPURACIÓN DEL AGUA MEDIANTE PROCESOS BIOTECNOLÓGICOS: LA BIOELECTROGÉNESIS

La bioelectrogénesis es un nuevo tipo de reacción biológica mediante la cual un microorganismo puede ceder o aceptar electrones directamente de un sólido conductor. Este proceso fue descubierto en 2003 [26] y, en 2008, dio lugar al diseño de las llamadas celdas de combustible microbianas (MFC) [27][3].

En los dispositivos MFC se utilizan microorganismos para oxidar el combustible, normalmente materia orgánica, y transferir los electrones producidos al ánodo. En su versión más simple, este electrodo está conectado al cátodo a través de un circuito eléctrico que se cierra con una resistencia. El ánodo y el cátodo están en contacto electrolítico a través de una membrana, y el flujo de especies cargadas entre ellas permite la producción de una corriente eléctrica.

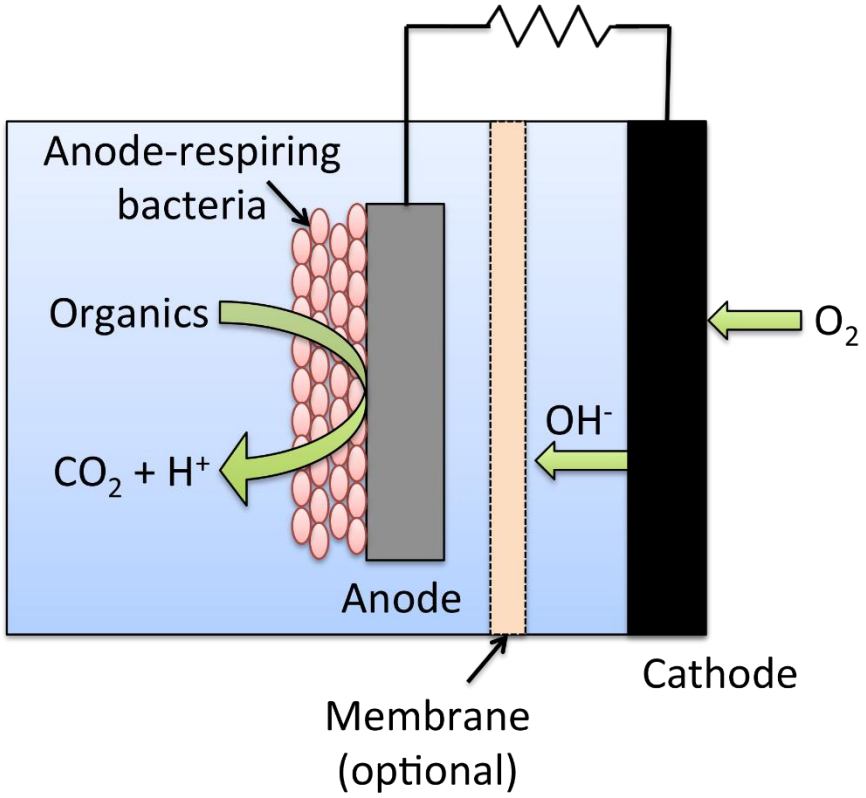

Ilustración 3: Funcionamiento de un MFC. Fuente: thevoltreport

Estos microorganismos, denominados electrogénicos, se comportan frente a los electrodos de grafito de los dispositivos MFC como si se trataran de los aceptores de electrones insolubles en sus hábitats naturales, lo que les permite oxidar la materia

orgánica y crecer a expensas de este proceso. El resultado final es la oxidación del combustible por medios naturales y la producción limpia de energía eléctrica.

2.2 REDES DE SENSORES INALÁMBRICAS

Las redes de sensores inalámbricas (WSN, del inglés ”Wireless Sensor Network”) son redes de dispositivos de tamaño mínimo (llamados nodos o “motas”) con capacidad de cómputo, almacenamiento y comunicación equipados con sensores de medición que se distribuyen geográficamente en un ambiente con el objetivo de monitorizarlo y la posibilidad de actuar sobre el mismo [.

El desarrollo de las WSN surgió a raíz de aplicaciones militares. Sin embargo, a día de hoy, su uso está orientado al campo de la industria y de las aplicaciones de usuario. Ejemplos de campos en los que están enfocadas las WSN son los siguientes:

- Monitorización de ambientes controlados: detección de incendios, contaminación del aire/agua, etc.
- Monitorización industrial: consumo de agua, aplicaciones en el sector agrícola, etc.
- Aplicaciones médicas
- Otras aplicaciones, tales como la domótica, seguridad, etc.

A continuación se explicarán algunos aspectos relevantes de las WSN para este proyecto: los elementos y estructura de este tipo de redes y la aplicación en la que está enmarcada este proyecto, que es la monitorización ambiental.

2.2.1 ELEMENTOS DE UNA RED DE SENSORES INALÁMBRICOS

Los elementos que conforman típicamente una WSN son los siguientes:

- **Nodos sensores (motas):** son los encargados de obtener los datos obtenidos por los sensores que llevan conectados/integrados, para después enviarlos a un nodo central.

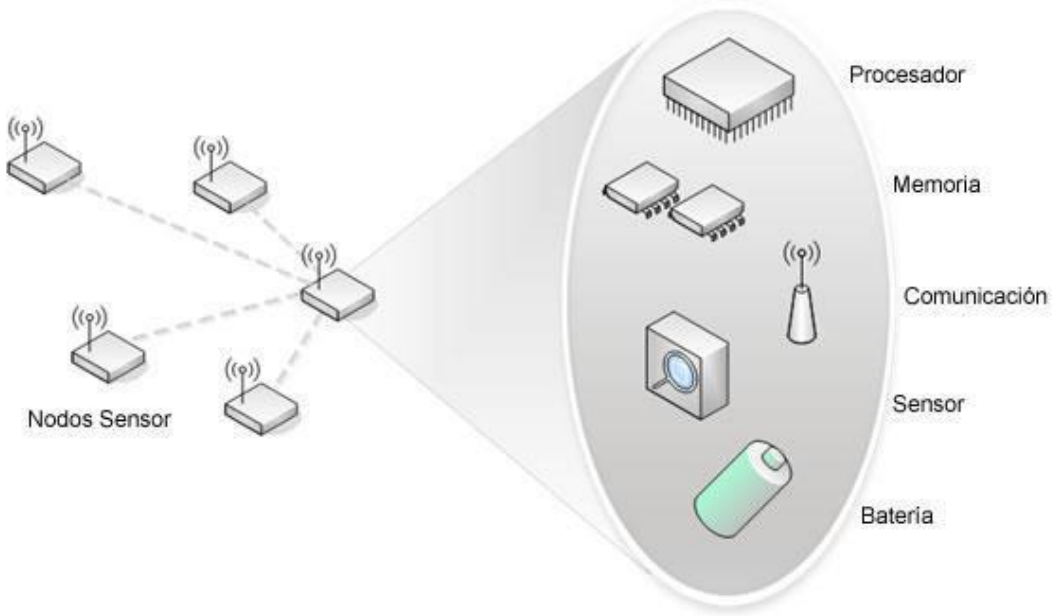

Ilustración 4: Componentes internos de un nodo sensor.

- **Sensores:** son dispositivos capaces de obtener medidas de magnitudes físicas o químicas. Los principales parámetros que caracterizan la calidad de un sensor son la sensibilidad, resolución y precisión, aunque existen más parámetros de calidad.
- **Nodo central (gateway):** es el nodo encargado de intercomunicar a la red de sensores con la estación base, enviando toda la información obtenida por la red para su posterior análisis/procesado.
- **Estación base:** es el elemento encargado de procesar los datos recibidos por la red de sensores. Típicamente el enlace entre la red de sensores y la estación base es a través de una red TCP/IP.
- **Red inalámbrica:** es el medio de comunicación dentro de la red de sensores. Se puede utilizar cualquier protocolo inalámbrico (WiFi, WiMax, Bluetooth). Sin embargo, el estándar 802.15.4 ZigBee está especialmente diseñado para este propósito, debido a su bajo consumo (el consumo en ZigBee es de 30 mA transmitiendo y de 3 µA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo de Bluetooth [referencia].

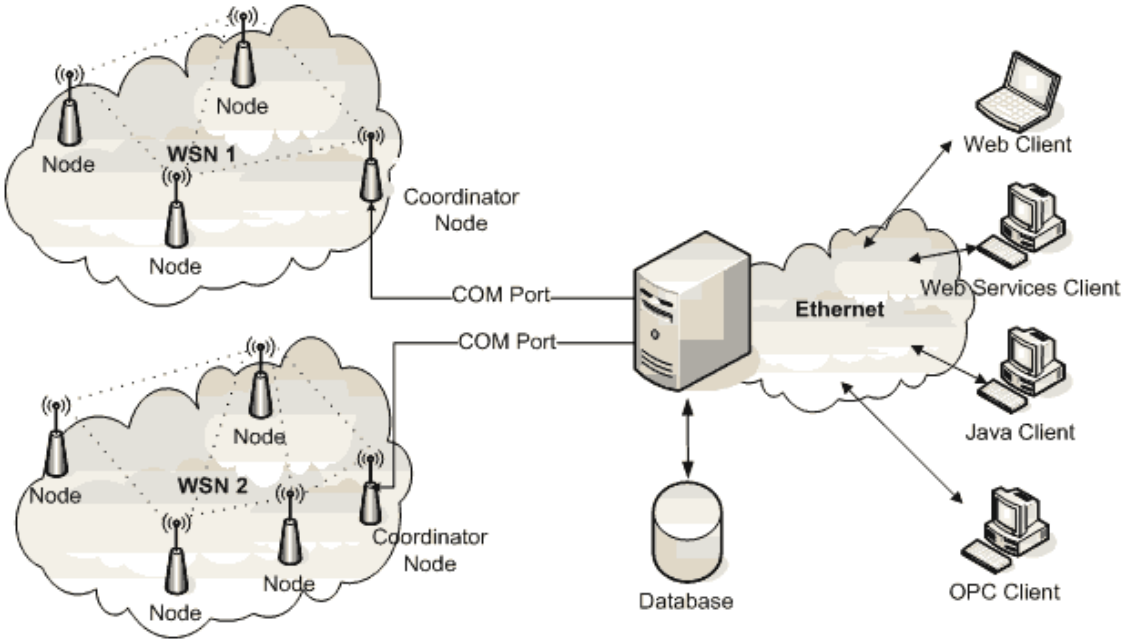

Ilustración 5: Escenario típico de una red WSN integrado con otros servicios. Fuente: http://www.iebmedia.com

2.2.2 APLICACIONES DE LAS REDES DE SENSORES INALÁMBRICOS: MONITORIZACIÓN DE AMBIENTES CONTROLADOS

Una de las aplicaciones más relevantes que se le puede dar a las WSN es la monitorización ambiental. Esta aplicación consiste en la medición u observación de ciertos parámetros en un determinado ambiente, con la finalidad de verificar si están ocurriendo determinados impactos ambientales y, en base a las observaciones, tomar las medidas oportunas.

Para el desarrollo de esta tarea, la metodología utilizada hasta ahora ha sido la de realizar mediciones de forma manual, desplegando personal en el medio de forma periódica y necesitando a alguien encargado de la monitorización de las mismas, sin poder realizar actuaciones en el sistema de forma automática. Por tanto, el agregar WSN a los sistemas de monitorización es un avance significativo capaz de solventar los problemas mencionados, ya que las WSN sólo deben ser desplegadas una vez, para después ser controladas de forma remota e incluso de forma automatizada.

Varios de los puntos a analizar a la hora de desplegar un sistema de monitorización ambiental son los siguientes:

- Los parámetros que van a ser medidos.
- La tecnología (sensores) que van a obtener estos datos.
- La topología de red óptima (debe ser estable para no tener que necesitar reconfiguraciones).
- La frecuencia de la muestro/obtención de datos.

Las WSN en este tipo de aplicaciones deben tener un tiempo de vida alto, ya que los fenómenos y parámetros a monitorizar tienen objeto de ser a largo plazo. Esto significa que uno de los parámetros a tener más en cuenta es el consumo eléctrico y vida de la batería. Los nodos de la WSN en este tipo de aplicación deben, además, estar altamente sincronizados. Otro aspecto a destacar es que no existen requerimientos de latencia estrictos.

Con esta red desplegada, se pueden conseguir objetivos tales como:

- Sistema de alarmas automático.
- Mapas sensoriales en tiempo real.
- Representación de gráficos e históricos.
- Elaboración de informes.
- Sistemas de actuación en el ambiente, bien automáticos o de forma remota vía Internet.

2.3 PROYECTOS RELACIONADOS

En el capítulo 1 de esta memoria se mencionó que la tarea desarrollada en este TFG está incluida dentro de un proyecto mayor, *Smart Wetland*. En esta sección se procede a describir la evolución de los proyectos que se han llevado a cabo hasta llegar al proyecto *Smart Wetland*.

2.3.1 PROYECTO ANTECEDENTE: AQUAELECTRA

El proyecto *Aquaelectra* es un proyecto de la convocatoria INNPACTO 2010 en el que participaron los siguientes miembros: el grupo Bioelectrogénesis de IMDEA-Agua, la Fundación CENTA y las empresas de tratamientos de aguas Euroestudios, JOCA y DAM.

Este proyecto perseguía el objetivo de la implantación de un sistema natural de depuración de aguas residuales mediante humedales bioelectrogénicos con la tecnología y avances sobre los que se habló en este mismo capítulo anteriormente.

2.3.1 PROYECTO ACTUAL: SMART WETLAND

El proyecto Smart Wetland surge como continuación del proyecto Aquaelectra, en la convocatoria INNPACTO 2012 del MINECO. Los integrantes de este nuevo proyecto son: la empresa EUROESTUDIOS, coordinadora del proyecto, el instituto IMDEA-Agua (Grupo Bioelectrogénesis), la Fundación Centro de las Nuevas Tecnologías del Agua (CENTA), el grupo de Instrumentación Avanzada del Centro de Astrobiología (CSIC-INTA) del Instituto Nacional Técnico Aeroespacial (INTA) y la empresa A-CING.

El objetivo de este proyecto es la continuación de las tareas investigadas y desarrolladas durante el proyecto anterior, añadiendo una red de sensores inalámbricos con la capacidad de modificar los parámetros que condicionan el proceso de depuración del agua. Además, este sistema de control también debe permitir ser controlado de manera remota mediante TIC (Tecnologías de la Información y la Comunicación).

Esta última tarea (la aplicación web que sirva como herramienta para el telecontrol del humedal) recae en la empresa A-CING, y es la tarea desarrollada en implementada en este TFG.

Dado que esta aplicación debe comunicarse de forma directa con la red de sensores inalámbricos encargada del control del humedal, a lo largo de esta memoria se hablará de algunos aspectos de la misma a pesar de no haber intervenido en los procesos de estudio, diseño e implementación, ya que se considera importante para la comprensión de la tarea desarrollada.

2.4 DESARROLLO DE APLICACIONES WEB DEL LADO DEL SERVIDOR

La programación en el lado del servidor consiste en todos los programas que se ejecutan en el servidor.

En el contexto de las aplicaciones web, el servidor es el encargado de las siguientes tareas:

- Procesar las los datos que el usuario envía a la página.
- Mostrar las páginas (responder a las peticiones HTTP del cliente, enviándole la página que está solicitando, en caso de que el cliente cumpla los requisitos).

- Procesos internos de la página, es decir, cualquier tarea interna que deba realizar la aplicación dependiendo de las especificaciones.
- Interaccionar con bases de datos, o cualquier otra estructura de almacenamiento donde se encuentren los datos que maneja la aplicación.

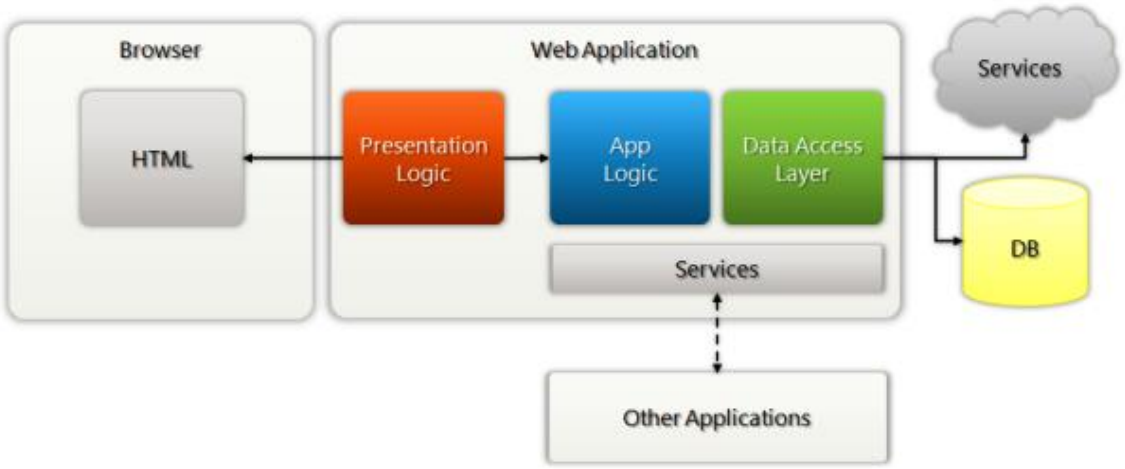

Ilustración 6: Posible arquitectura de una aplicación web.

Para el desarrollo de esta tarea existen multitud de lenguajes de programación y tecnologías, tales como PHP, ASP.net, C++, Java, Python, etc. En este apartado se describirán las tecnologías utilizadas en el proyecto con este fin.

2.4.1 LENGUAJES DE PROGRAMACIÓN PARA EL DESARROLLO DE APLICACIONES WEB: PYTHON

Python es un lenguaje de programación de alto nivel, con una sintaxis sencilla y fácil de aprender.

Fue desarrollado a finales de los ochenta por Guido van Rossum en Países Bajos. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License [23], que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores. Por tanto, su uso es libre incluso con fines comerciales [14].

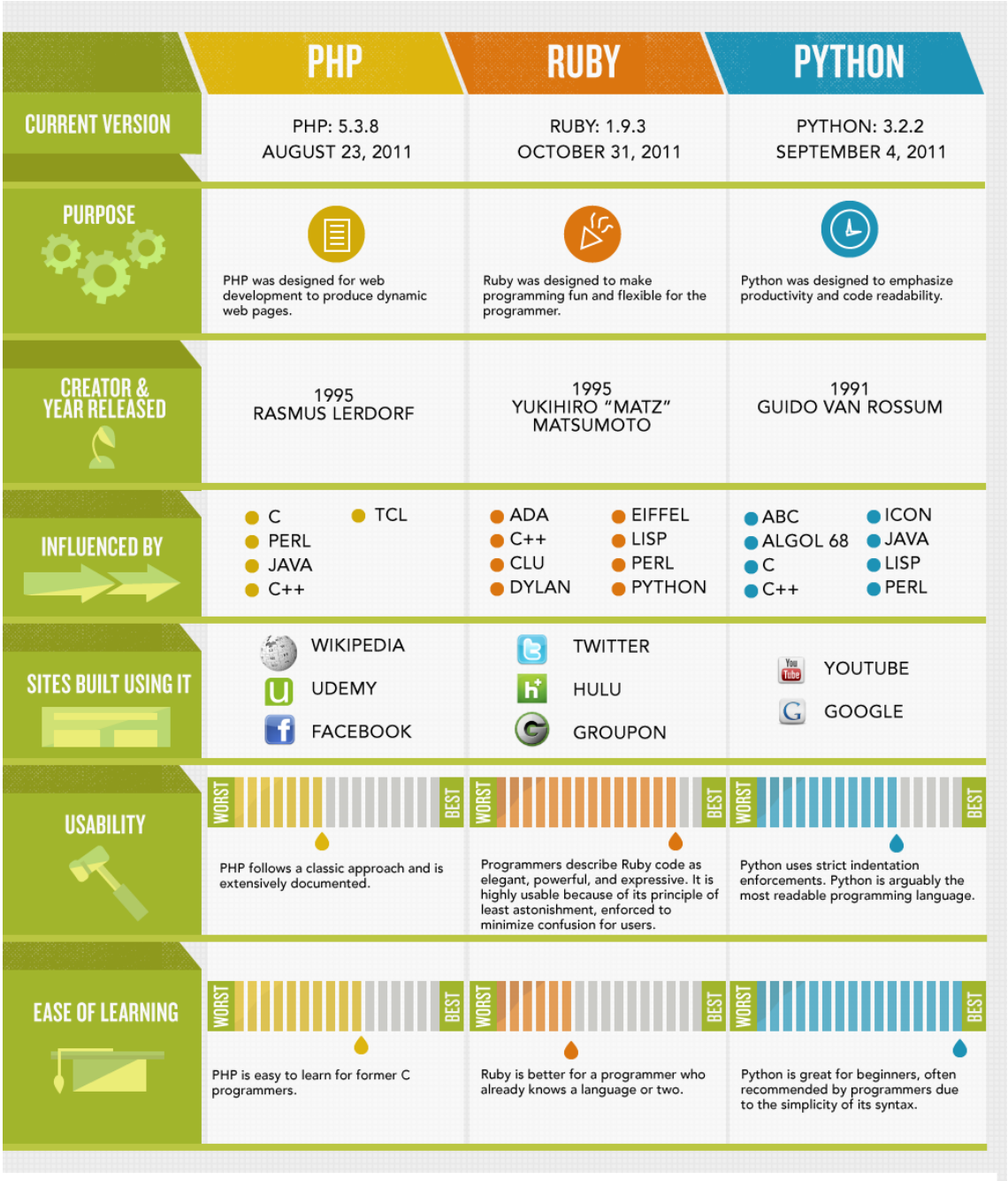

Ilustración 7: Comparación de lenguajes de programación modernos. Fuente: udemy.com

Python es uno de los lenguajes de programación de código abierto más comunes y populares. Como resultado, hay una gran cantidad de conocimientos a disposición de los programadores de Python y es apoyado por una amplia comunidad de desarrolladores. Gracias a ello, la cantidad de librerías y tutoriales que existen es muy extensa, lo que contribuye a que Python crezca y mejore continuamente.

Algunas de las características que convierten a Python en un buen lenguaje para el desarrollo de software son las siguientes:

- Soporta orientación a objetos, lo que es útil a la hora de reutilizar y extender código, permite crear sistemas más complejos (adaptándose de forma más fácil a problemas reales), permite crear sistemas más robustos y agiliza enormemente el desarrollo de software.
- Es un lenguaje de alto nivel, lo que permite no tener que preocuparse por detalles de bajo nivel (como por ejemplo, manejar la memoria utilizada por el programa), y así centrarse únicamente en la solución del problema.
- Es un lenguaje de programación interpretado, lo que quiere decir que no es necesario compilar y ejecutar un programa en Python, como se haría por ejemplo en otros lenguajes como C, C++ o Java.
- Los programas escritos en Python son portables. Gracias al punto anterior, se consigue que cualquier programa escrito en Python pueda ser utilizado en cualquiera de las plataformas compatibles, a excepción de las librerías compatibles sólo con alguna plataforma en concreto.
- La librería estándar de Python es muy amplia y contiene funciones que permiten realizar una gran cantidad de tareas: manejar expresiones, generación de documentos, realización de pruebas, manejar procesos, bases de datos, web, GUI (Graphical User Interface), etc. Sin embargo, uno de los puntos fuertes de Python son las librerías externas, que se pueden adaptar mejor a las necesidades de un proyecto, o realizar tareas que no se pueden realizar con la librería estándar, como por ejemplo: operaciones con algunas bases de datos, comunicación con puertos serie, etc.

2.4.2 FRAMEWORKS PARA EL DESARROLLO DE APLICACIONES WEB EN PYTHON: DJANGO

Django es un *framework* para desarrollo web de código abierto escrito completamente en Python.

Fue desarrollado originalmente para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005. Django es gestionado, desde 2008 hasta el presente por la Django Software Foundation. [8][10]

Django promueve el principio DRY (Don't Repeat Yourself), para evitar que los programadores repitan fragmentos de código y funciones que ya han escrito en otras

partes de la aplicación. Para ayudar a cumplir este objetivo, proporciona una gran cantidad de funciones y atajos para tareas repetitivas, tal y como se explicará en el resto de esta sección.

Tanto la documentación que existe sobre Django, como la comunidad activa de desarrolladores son tremendamente amplias, lo que facilita enormemente el aprendizaje y la solución de los errores más comunes que puedan aparecer en las tareas de desarrollo.

A lo largo de esta se explicarán diferentes aspectos relevantes del *framework*, tales como sus características generales, la arquitectura que sigue en diferentes ámbitos o algunas de sus características especiales, como son las opciones de seguridad que ofrece.

2.4.2.1 Características generales

Un *framework* web tiene el objetivo de agilizar el proceso de construcción de sitios web complejos y dinámicos, además de ofrecer funciones y atajos para las tareas de programación más comunes y repetitivas.

Al ser un *framework* completamente escrito en Python, este posee las características que ofrece el lenguaje de programación Python. Una de las más importantes y ventajosas a la hora de diseñar aplicaciones web, es que Python es un lenguaje de programación de alto nivel. Por lo tanto permite abstraerse de los detalles de bajo nivel para centrarse directamente en cumplir las especificaciones del sitio web.

Otra de las ventajas de Python a la hora de programar sitios web, es que Python sea un lenguaje de programación interpretado. Esto supone no tener que recompilar y ejecutar todo el código cada que vez que hagamos una modificación en la página, ahorrando así una cantidad de tiempo considerable.

Además, Django sigue la arquitectura MTV (model-template-view), separando las distintas tareas en el desarrollo de aplicaciones web de forma clara y eficaz. Esto facilita el trabajo conjunto entre diseñadores y programadores, ya que al estar separado el código, estos no interfieren entre sí a la hora de hacer cambios y trabajar de forma separada.

A continuación se describen algunas de las principales herramientas del *framework*.

Mapeo de urls

Proporciona una herramienta para mapear URLs con el código que se ejecuta al realizarse la petición. También se pueden asignar patrones de URL, en vez de tener que asociar las URLs al método que atiende las peticiones de forma estática. De esta manera, el programador puede elegir exactamente cómo son las URLs de la aplicación web, al contrario que con otros *frameworks*/lenguajes de programación. Esto es ventajoso por múltiples motivos. El motivo más lógico es que es más "amigable" para el usuario, ya que son URLs legibles, y por tanto se pueden recordar con mayor facilidad. Otro motivo es que los buscadores como Google encontrarán con mayor facilidad URLs con un formato más simple.

Django admin

Django proporciona al programador una interfaz de administración generada de forma automática, que permite realizar gran variedad de operaciones sobre los objetos (en django llamados "modelos") almacenados en la base de datos (incluyendo usuarios de la página): leer, actualizar, crear, eliminar, validar, gestionar la relación con otros objetos, etc., llevando además un registro de todas las acciones realizadas.

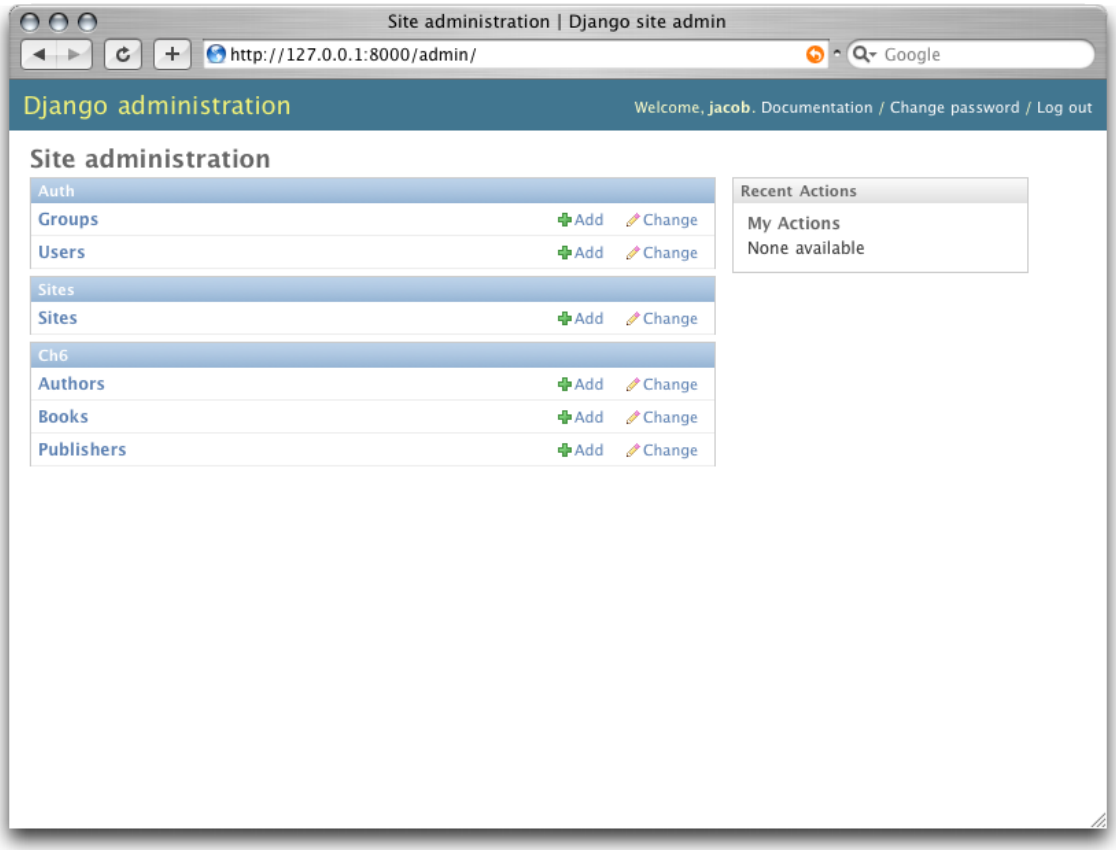

Ilustración 8: Página principal de la herramienta de administración de Django.

Formularios HTML

Automatiza la tarea de manipulación de formularios HTML, tanto como para validar los datos (que se ajusten a las restricciones que posee un campo en concreto), como para mostrarlos o convertir los tipos de datos del formulario a los de la aplicación.

Django ofrece un sistema de traducción de sintaxis de lenguajes como SQL, HTTP o XML a Python/Django. De esta manera, se pueden hacer consultas a la base de datos sin tener que escribir la consulta SQL como cadena de texto, o formar peticiones HTTP de forma automática, simplemente enviando los campos que se desee rellenar como parámetros a las funciones que proporciona Django para realizar esta tarea.

En la documentación oficial de Django **[https://docs.djangoproject.com/en/dev/ref/models/querysets/#ref-models-querysets]** existen ejemplos y se especifica el uso de todas las funciones que ofrece Django para manejar y manipular objetos de la base de datos. También se explica todo

lo relacionado con las peticiones HTTP y el módulo HTTP que ofrece Django para tal finalidad [https://docs.djangoproject.com/en/dev/ref/request-response/]

Entorno de desarrollo

Django ofrece un entorno de desarrollo muy completo con la herramienta manage.py, que se crea por defecto al iniciar un proyecto en Django, o la debug toolbar [https://github.com/django-debug-toolbar/django-debug-toolbar], un módulo externo muy útil para ver, por ejemplo, las consultas SQL que se están realizando en tiempo real mientras se navega por la página.

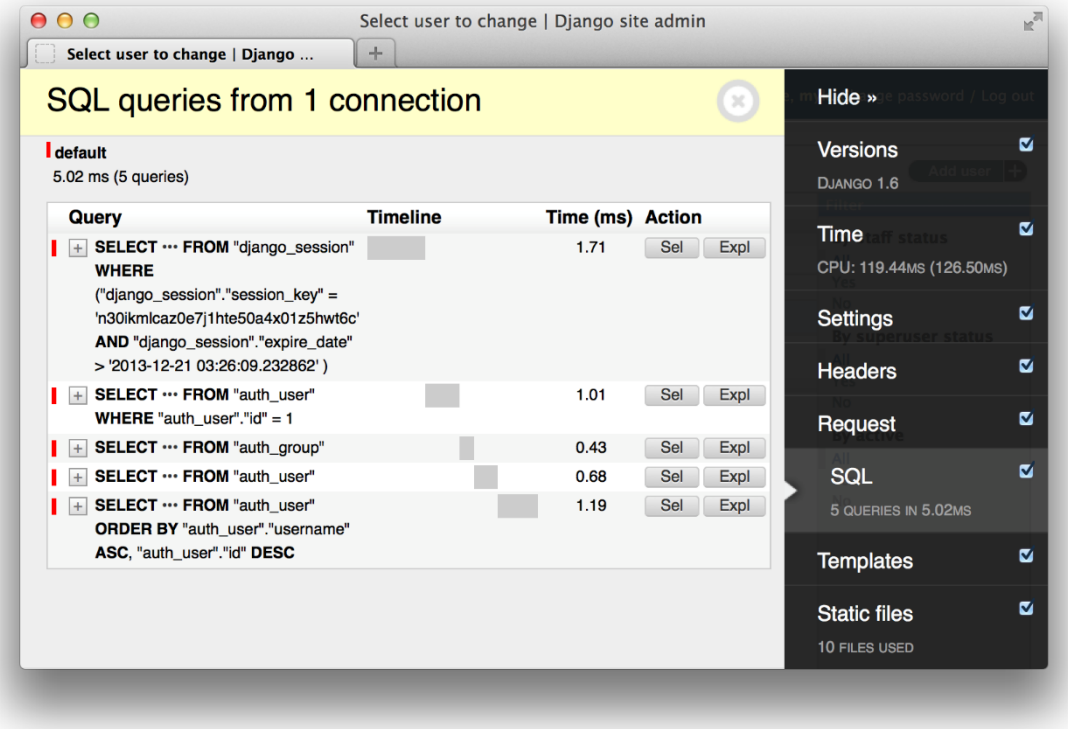

Ilustración 9: Uso de Django Debug Toolbar para ver consultas SQL en tiempo real.

La herramienta manage.py provee al desarrollador de diferentes funcionalidades, tal como se indica en la documentación oficial [24]. Las más importantes son las siguientes:

- Con el comando dbshell, se abre una consola para consultar la base de datos asociada al proyecto, con los parámetros de usuario, contraseña y otras configuraciones especificadas en el módulo settings.py.

- Con runserver se ejecuta el servidor de desarrollo de Django, un servidor web muy ligero escrito completamente en Python, que permite a los programadores ver los cambios realizados en la web de manera rápida, sin tener que meterse en la configuración de un servidor de producción (como por ejemplo Apache), hasta que no sea necesario, lo que supone un ahorro de tiempo considerable.
- Con startapp se crea el esqueleto de una aplicación de un proyecto de Django de forma automática (todos los .py obligatorios en una app en Django, la estructura, etc.).
- Django provee al programador de una herramienta para realizar pruebas automatizadas de todas las funciones del código de la aplicación. Esto puede ser útil para probar que el código que es escrito se comporta del modo esperado, tanto como para las nuevas funciones, tanto como para las modificaciones de las ya existentes. Esta herramienta permite, además de enviar datos de prueba como entrada a una función, simular peticiones HTTP y comprobar que el comportamiento de la aplicación es el adecuado.

2.4.2.2 Arquitectura de Django

Como se ha comentado anteriormente, Django sigue la arquitectura MTV (Model-Template-View). Esta arquitectura es muy parecida a la convencional MVC (Model-View-Controller), ya que ambas separan los datos y la lógica de negocio de la interfaz de usuario y las partes que gestionan los eventos y las comunicaciones, con algunas diferencias de nomenclatura:

- El modelo es igual en ambas arquitecturas.
- El controlador, en Django, corresponde a la vista.
- La vista, que muestra los datos al usuario, en Django se corresponde con las plantillas.

A continuación se definirán las funciones que realiza cada uno de los elementos que conforman la arquitectura MTV.

El modelo

El modelo define la lógica de negocio de la aplicación, es decir, todas las operaciones que manejan los datos de la aplicación, tales como entradas o modificación de datos, consultas a los datos, y todo el procesamiento que se realiza detrás de la aplicación de

forma invisible al usuario. Todo esto permite separar el estado y la forma en la que se manipulan los datos del resto de la aplicación.

En Django, esto se consigue a través de un mapeo objeto-relacional (ORM), en el que se mapean clases de Python con estructuras físicas de datos, tales como bases de datos SQL, XML, ficheros de texto, etc. El módulo de Python que define el modelo en la aplicación es models.py

La vista

Las vistas son las funciones que se encargan de la comunicación entre las diferentes entidades de la aplicación. Las vistas atienden las peticiones HTTP, interactúan con el modelo, realizan las operaciones necesarias con los datos y después envían el resultado a las plantillas. El módulo donde se definen las funciones de la vista se llama views.py.

Django provee de funciones para manejar peticiones HTTP de forma sencilla y adaptada a las circunstancias de la aplicación. Por ejemplo, ofrece atajos para retornar páginas de error 400, diferentes funciones para representar los datos que envía a la plantilla, manejo de cookies, etc. También ofrece funciones relativas al manejo y autenticación de usuarios en la aplicación.

La plantilla

Django posee un lenguaje para incrustar dentro de las plantillas HTML, que son denominados template tags. Este elemento de Django permite introducir código y funciones de Python/Django dentro de las plantillas HTML, para así añadir lógica a la página (por ejemplo, mostrar el nombre del usuario en caso de estar autenticado y en caso de no estarlo, mostrar un link para registrarse en la página).

Funcionamiento de la arquitectura MTV

La comunicación e interacción entre los distintos elementos que forman la arquitectura de una aplicación web en Django cuando una URL es solicitada es la siguiente:

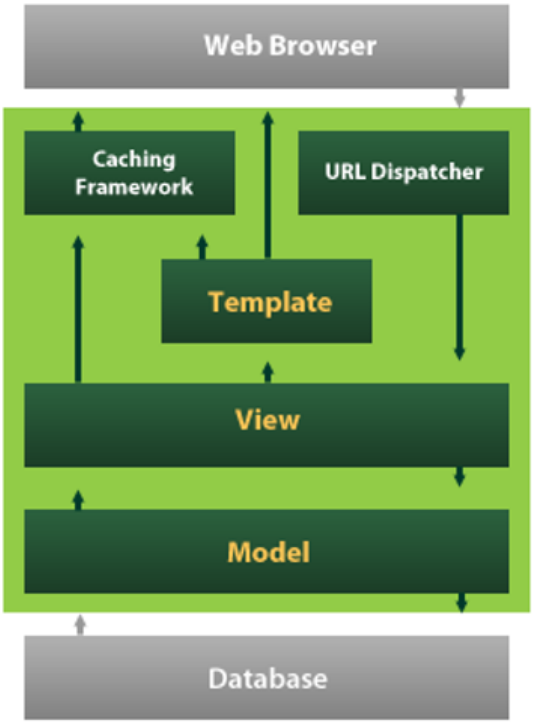

Ilustración 10: Arquitectura de Django. Fuente: djangobook.com

1. La herramienta de mapeo de URLs (sobre la que se habló anteriormente en el apartado "Características generales") llama a la vista que corresponde a la URL solicitada. Si la función de caché de Django está activada (esta funcionalidad se puede activar/desactivar de forma individual en las diferentes páginas de la aplicación web, es decir, el hecho de que esté activada no implica que se vaya a hacer caché en toda la aplicación), se comprueba si la página está almacenada en caché y, en caso positivo, se devuelve.
2. La función de la vista recibirá una petición HTTP, que normalmente implicará acceder a los datos de la aplicación, representados por el modelo. A su vez, el modelo, representará los datos de la aplicación comunicándose con el sistema de almacenamiento existente (generalmente una base de datos).
3. La plantilla recibirá los datos de la respuesta provenientes de la vista. En las plantillas se procesarán estos datos para generar la página HTML correspondiente.
4. Finalmente, la vista retornará una respuesta HTTP, conteniendo la página HTML representada, habiendo realizado en el proceso las operaciones oportunas.

Jerarquía de ficheros en un proyecto Django

En Django se diferencia entre proyecto y aplicación. La diferencia entre ambos es la siguiente:

- Una aplicación debe contener un conjunto de vistas, modelos y plantillas relacionadas entre sí, lo más independiente posible del resto de aplicaciones, con el fin de aportar una funcionalidad a la aplicación web.
- Un proyecto es un conjunto de aplicaciones que utilizan la misma configuración, lo que significa usar la misma base de datos/almacenamiento físico y otras características comunes. También es el que contiene el fichero urls.py principal de la aplicación. Cabe destacar que normalmente un proyecto se corresponde con un sitio web, pero no necesariamente (un sitio web puede estar conformado por varios proyectos Django).

Cuando utilizamos el comando django-admin startproject "nombre_proyecto" se crea una estructura general para el proyecto por defecto, así como los ficheros obligatorios para el mismo: __init__.py, manage.py, settings.py, urls.py y wsgi.py. Al crear una aplicación (con manage.py startapp), se crean los siguientes módulos para la misma: models.py, views.py, tests.py, admin.py e __init__.py. Sólo los módulos __init__.py y models.py son obligatorios para que no existan errores.

El módulo settings.py es donde se configura el proyecto. En este módulo se establecen configuraciones tales como la base de datos a usar, las direcciones de algunos directorios importantes para el proyecto, etc.

A modo ilustrativo, un árbol de ficheros de un proyecto Django de ejemplo sería el mostrado en la Ilustración 11:

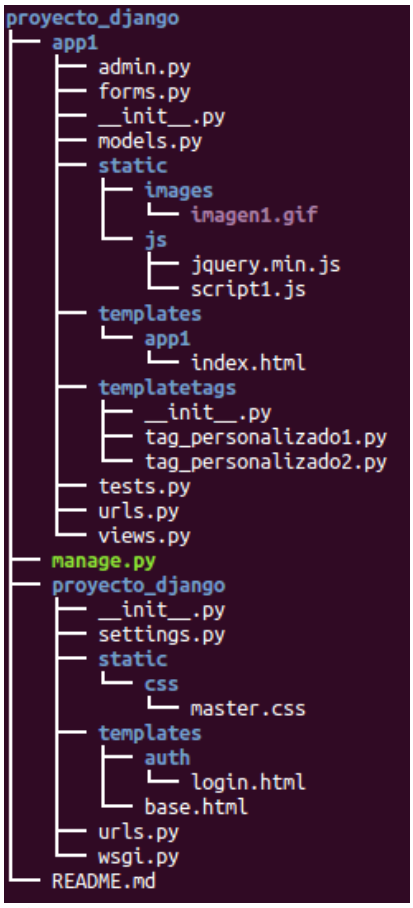

Ilustración 11: Árbol de ficheros de un proyecto en Django.

2.4.2.3 Seguridad por defecto en Django

En esta sección se procederá a describir algunos de los ataques de seguridad en aplicaciones web más típicos y de cómo Django ofrece seguridad frente a ellos de forma automática.

Inyección SQL

La inyección SQL es un tipo de ataque en el que un usuario malicioso introduce consultas SQL en alguna de las entradas de la aplicación web (por ejemplo en formularios HTML), con el fin de obtener información sensible de la base de datos o incluso borrar/modificar información. Se produce principalmente cuando no se validan las entradas del usuario correctamente y se hace la consulta SQL directamente con la cadena que ha introducido el usuario.

Django provee de varias herramientas para solucionar esto. La primera consiste en dejar a Django que forme las consultas SQL mediante su API para bases de datos. La

segunda opción sería, utilizando consultas SQL formadas por el programador, realizar la ejecución de la consulta con la API de Django. Para mayor información de cómo Django realiza esta funcionalidad se puede consultar. [7]

Cross-Site Scripting (XSS)

Esta vulnerabilidad consiste, de forma resumida, en que un usuario malicioso, a través de una URL, consigue ejecutar código JavaScript (u otro lenguaje de scripting) en el navegador de un tercero.

Por ejemplo, una web que realice el siguiente proceso:

1. La URL a su home tendrá la siguiente estructura: http://www.ejemplo.com/home/?nombre=Rodrigo
2. La aplicación cogerá el valor del atributo nombre para mostrarlo por pantalla.

Si una tercera parte modifica la URL y en vez de enviar como valor del atributo nombre, envía un script que se ejecute en el navegador, podría conseguir cosas tales como enviar información de la sesión del usuario y así poder suplantar su identidad o robar sus datos de la víctima que activó el hipervínculo.

Django provee una herramienta para evitar esto (aunque nunca al 100%), y es mediante el sistema de plantillas. Si Django recibe un parámetro en la petición HTTP con el formato <script> o similar, lo que escribirá al crear la plantilla HTML será <script>

Para mayor información, se puede consultar la documentación de Django. [7]

Cross Site Request Forgery (CSRF)

Esta vulnerabilidad típica en las aplicaciones web, consiste, al contrario que en XSS (el usuario confía en el sitio web, en la URL que hace click), cuando un sitio web confía en un usuario (un usuario autenticado, por ejemplo), y un tercero ejecuta una acción a través de este.

Por ejemplo, el atacante podría enviarle una URL modificada del sitio en el que la víctima esté autenticada, que borre algún elemento de la base de datos, u obtenga datos propios del usuario autenticado para enviárselos al atacante.

Django tiene una solución a este problema, que es el CSRF token. Viene activado en la configuración por defecto de Django. Consiste en un token que Django almacena en una sesión de usuario, y debe ser enviado en todas las peticiones HTTP que reciba y coincidir. Si no coincide, devolverá un mensaje de error al usuario.

Protección contra otras vulnerabilidades

Django proporciona protección frente a otras vulnerabilidades, explicadas en la documentación oficial [7], además de las 3 principales mencionadas anteriormente. Algunas de estas protecciones adicionales son:

- Opción para habilitar SSL (HTTPS)
- Protección contra *clickjacking*. [5]
- Validación de cabeceras de host.
- Seguridad en sesiones de usuario
- Seguridad en la subida de contenido por el usuario

2.4.3 APLICACIONES WEB EN TIEMPO REAL: NODE.JS Y SOCKET.IO

2.4.3.1 Node.js

Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación Javascript con una arquitectura orientada a eventos, cuya misión es permitir al programador construir servidores web altamente escalables y escribir código que maneje una gran cantidad de conexiones simultáneas en sólo una máquina física. Fue creado por Ryan Dahl en 2009 y actualmente es gestionado por la empresa Joyent. [22]

Node.js utiliza el motor de JavaScript V8 de Google [29], una VM de gran capacidad y velocidad y capacidad de I/O (Input/Output) de datos muy potente. Además, Node.js soporta protocolos de red tales como TCP, DNS y HTTP.

La arquitectura que sigue Node.js, al ser JavaScript, es la programación basada en eventos. Normalmente, al utilizar JavaScript en el lado del cliente, se atiende eventos tales como un click en algún elemento de la aplicación, al cargar la página, etc. En Node.js, los eventos que se escuchan son, por ejemplo, conexiones de clientes, mensajes recibidos, errores que no permiten continuar al programa, etc. Este modelo, aplicado a un servidor, es altamente escalable y útil para gran cantidad de funcionalidades que se necesitan en las aplicaciones web en la actualidad.

La manera en que funciona Node.js y que permite atender una gran cantidad de peticiones, es mediante un Bucle de Eventos. Node.js escucha peticiones en el bucle, y las operaciones de I/O de datos (como por ejemplo, consultas a bases de datos), que requieren mayor cantidad de recursos de la máquina y bloquean archivos se realizan de forma asíncrona. Cuando esta operación termina, se retornan los resultados al Bucle de Eventos. De esta manera, el bucle va delegando todas las tareas en hilos diferentes mientras sigue atendiendo nuevas peticiones.

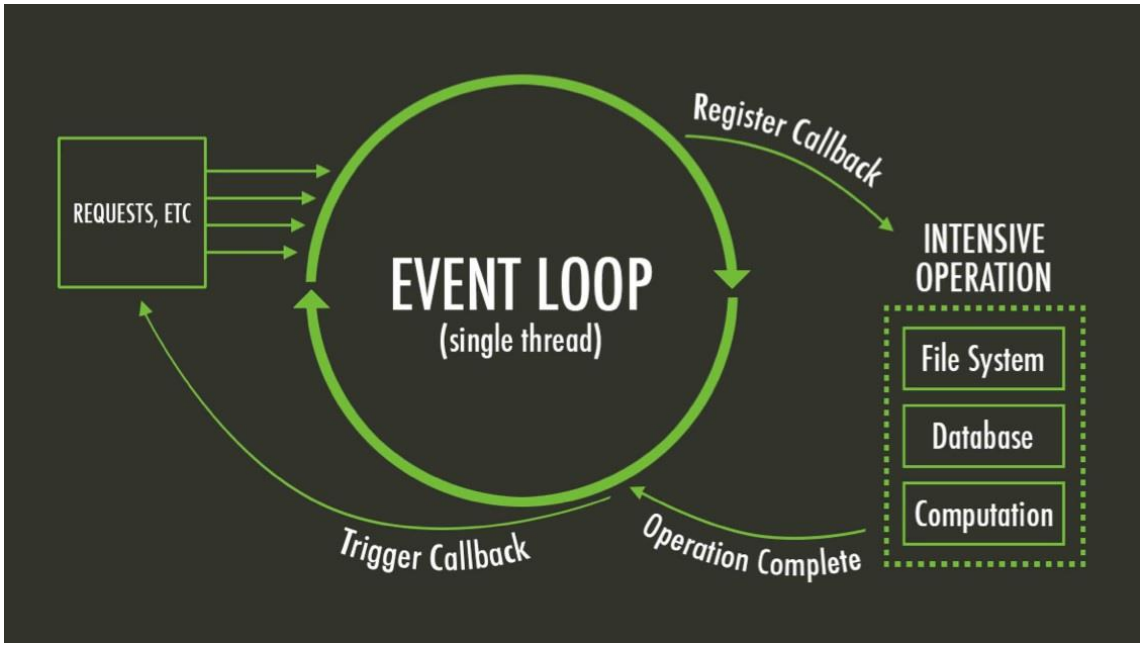

Ilustración 12: Esquema gráfico del Bucle de Eventos en Node.js.

Todo esto supone grandes ventajas, ya que con esta arquitectura, no se crea un hilo en el sistema operativo para atender cada petición, como hacen otros servidores, sino que las peticiones se atienden en un único hilo del sistema operativo (el Bucle de Eventos).

Además, gracias a la arquitectura basada en eventos, el servidor se "duerme" mientras no está actuando, lo que hace que se reduzca el consumo de recursos en la máquina del servidor drásticamente. El sistema operativo notifica al servidor Node cuando llega alguna petición. Cada conexión es sólo una pequeña porción de memoria en el sistema.

El potencial de Node.js es muy alto, además de que su sintaxis es relativamente sencilla y los programas escritos con esta tecnología se entienden con relativa facilidad. Además, ya que la mayoría de programadores web ya conocen JavaScript,

fundamental a la hora de programar aplicaciones web, la curva de aprendizaje es baja por lo general. Sin embargo, los servidores Node.js suelen tener una estructura excesivamente piramidal, debido a la arquitectura de eventos (se pueden anidar múltiples escuchadores de eventos en otros), y debido a esto puede ser complicado, si no se escribe de una forma ordenada, el entender el código.

Node.js, aparte de los paquetes que trae por defecto, tiene una gran cantidad de paquetes que implementan diferentes funcionalidades. Gracias a que la comunidad de desarrolladores que trabaja con Node.js es enorme, la cantidad de paquetes y nuevas funcionalidades que estos incluyen es creciente.

A continuación se hablará de la herramienta que utiliza Node.js para gestionar sus propios paquetes (npm), y de algunos de los paquetes fundamentales para la construcción de aplicaciones web en tiempo real.

2.4.3.2 Gestor de paquetes de Node.js: npm

Node.js tiene un gestor de paquetes (o también llamados “módulos”) oficial llamado npm.

Node.js prácticamente nunca se utiliza sólo, sino que se utilizan módulos adicionales con distintos fines. Algunos de los módulos más utilizados son los siguientes:

- Socket.io, un *framework* que implementa el protocolo de websockets, utilizado generalmente para la creación de servidores en tiempo real.
- Request, para realizar peticiones HTTP de una forma simple.
- Redis, un servidor de caché de memoria que conoce la estructura de los datos que alberga.
- Cookie, para manejar cookies HTTP.
- Mongoose, para manejar objetos de bases de datos MongoDB.
- Mysql, para integrar servidores Node.js con Mysql
- Forever, una herramienta que permite asegurarte que un servidor/script/programa siempre se ejecute aunque se de alguna excepción que haga que el programa pare.
- Less,una herramienta para escribir código CSS más consistente.
- Yeoman, una herramienta que genera el esqueleto básico de nuestra aplicación web de una manera eficaz e instala paquetes y herramientas útiles para los desarrolladores de forma automática, como por ejemplo JQuery, Twitter Bootstrap o Modernizr, entre otros.

- Grunt, un ejecutador de tareas basado en JavaScript
- Bower, otro gestor de paquetes JavaScript

El algoritmo que utiliza npm para buscar los paquetes dentro de nuestro sistema de ficheros es el siguiente: cuando se hace un require('nombre_módulo') dentro de nuestro servidor Node, se busca en el mismo directorio una carpeta llamada node_modules, y dentro de ella busca el nombre del módulo. Si no encuentra la carpeta, sigue bajando niveles dentro del sistema de ficheros hasta que lo encuentre, y si finalmente no encuentre ningún directorio node_modules o no está el módulo especificado en este directorio, devuelve un error.

2.4.3.3 Socket.io

Socket.io es una librería de JavaScript utilizada para crear aplicaciones web en tiempo real. Está compuesta por un cliente, que se ejecuta directamente en el navegador, y por un servidor, una librería (módulo) para Node.js.

Tradicionalmente, para crear aplicaciones web en tiempo real, las alternativas han sido las siguientes (por orden de eficiencia):

- Polling asíncrono de datos cada X tiempo mediante AJAX.
- Manteniendo la conexión con el servidor abierta mediante el uso de XMLHttpRequest.
- Mediante el uso de websockets, una tecnología diseñada para conseguir comunicación bidireccional a través de HTTP.

La desventaja de Websocket es que no está soportado por todos los navegadores, además de que el servidor debe implementar características de seguridad especiales de Websockets (es decir, el servidor debe adaptarse al protocolo).

Socket.io soluciona este problema, ya que, además de ser un servidor Node.js basado en eventos, con las ventajas que esto conlleva (tal y como se explicaba en el apartado de Node.js), soporta 6 tipos de transporte, que son los siguientes:

- WebSocket
- Adobe® Flash® Socket
- AJAX long polling
- AJAX multipart streaming
- Forever Iframe

- JSONP Polling

Por tanto, sea cual sea el navegador o servidor que estemos utilizando, socket.io se adaptará y utilizará la tecnología para el transporte soportada/óptima.

2.4.4 DESPLIEGUE DE APLICACIONES WEB EN EL SERVIDOR: NGINX Y GUNICORN

2.4.4.1 Nginx

Nginx es un servidor web ligero de alto rendimiento. Es gestionado por NGINX, Inc., que fue fundada en Julio de 2011 por Igor Sysoev, Jefe Arquitecto del proyecto Nginx [http://www.linuxjournal.com/article/10108]. Está licenciado bajo la licencia BSD [https://www.gnu.org/licenses/license-list.html#OriginalBSD] y está soportado en los siguientes OS: Linux, OS basados en BSD, Mac OS X, Solaris, AIX, HP-UX y Windows Server 2003.

Ilustración 13: Logo de Nginx. Fuente: freesoftwaremagazine.com

Nginx sigue una arquitectura basada en eventos asíncrona para manejar peticiones, en vez de la arquitectura basada en hilos/procesos que sigue Apache, donde se necesitan tecnologías adicionales para atender procesos de forma asíncrona. Esto permite que Nginx sea más robusto y que tenga mayor rápidez que Apache en entornos con gran carga.

También posee un proxy inverso, lo que mejora las prestaciones de seguridad/rendimiento del servidor web.

Un resumen de las funcionalidades que ofrece el servidor web Nginx son las siguientes:

- Soporta SSL/TLS.
- Distribución de carga entre servidores gracias al proxy inverso.

- Caché de ficheros estáticos (imágenes, ficheros javascript, plantillas css, etc.).
- Compresión de archivos.
- FastCGI (protocolo utilizado para reducir la carga asociada a la interconexión entre el servidor web y las aplicaciones que lo soporten [http://www.nongnu.org/fastcgi/#AEN147]).
- Control de acceso.

2.4.4.2 Gunicorn

Gunicorn es un servidor web para Python y sistemas Unix. Sigue el modelo de multiprocesamiento Prefork, que consiste en crear diferentes procesos independientes para atender diferentes peticiones, además de soportar WSGI para la interconexión de aplicaciones Python y Gunicorn. [11]

Ilustración 14: Logo de Gunicorn. Fuente: gunicorn.org

Es un servidor web implementado de forma simple y compatible con multitud de *frameworks* (incluido Django). Además, consume una baja cantidad de recursos en el servidor (CPU y RAM) ofreciendo a su vez una buena velocidad.

2.5 DESARROLLO DE APLICACIONES WEB DEL LADO DEL CLIENTE

La programación en el lado del cliente se refiere a todos los programas que se ejecutan en la máquina del cliente.

En el contexto de las aplicaciones web, el cliente es el usuario a través de un navegador web, que es capa, de enviar y recibir peticiones HTTP e interpretar los resultados.

Este tipo de programación se utiliza con diferentes fines. Algunos de ellos son:

- Para hacer webs dinámicas, es decir, algún aspecto de la web cambia sin necesidad de solicitar páginas al servidor (esto es, recargar la página).

- Para enviar peticiones al servidor y recibir los datos contenidos en las respuestas.
- Para hacer páginas web interactivas, como por ejemplo, arrastrar y soltar un elemento de la página y que algo suceda, listas desplegables, etc.
- Para interaccionar con algún tipo de almacenamiento temporal del navegador, como por ejemplo las cookies.

Para el desarrollo de programas que se ejecutan en el cliente se utiliza principalmente el lenguaje de programación javascript y HTML/CSS. A continuación se describirán las tecnologías utilizadas en el proyecto con este fin.

2.5.1 LENGUAJES DE SCRIPTING EN EL NAVEGADOR: JAVASCRIPT

JavaScript es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript].

Se utiliza principalmente para la programación del lado del cliente, aunque existe una forma de JavaScript del lado del servidor (tal y como se habló en la sección 2.3.4 de este mismo capítulo).

El uso más común de JavaScript es escribir funciones incluidas en páginas HTML y que interactúan con el Document Object Model (DOM o Modelo de Objetos del Documento) [28] de la página. Algunos ejemplos de este uso son:

- Cargar nuevo contenido para la página o enviar datos al servidor a través de AJAX sin necesidad de recargar la página.
- Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.
- Contenido interactivo, por ejemplo, juegos y reproducción de audio y vídeo.
- Validación de los valores de entrada de un formulario HTML para asegurarse de que son válidos antes de ser enviado al servidor.
- Transmisión de información sobre los hábitos de lectura de los usuarios y las actividades de navegación a varios sitios web.

Dado que el código JavaScript puede ejecutarse localmente en el navegador del usuario (en lugar de en un servidor remoto), el navegador puede responder a las acciones del usuario con rapidez, haciendo una aplicación más sensible y dinámica.

2.5.2 INTERCAMBIO DE DATOS EN JAVASCRIPT: JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de los objetos de JavaScript, lo que permite no tener que depender de XML para el intercambio de datos.

JSON está construido en 2 estructuras:

- Una colección de nombre->valor, tal como los diccionarios en Python.
- Una lista ordenada de valores, tal como los array en muchos lenguajes de programación.

Un ejemplo de un mensaje en formato JSON, y ese mismo formato en XML sería el siguiente:

```
Autor = {"autor": {
  "nombre": "Rodrigo",
  "apellidos": [{"apellido": "Argüello"},{"apellido": "Flores"}],
  }
}


<?xml version="1.0" encoding="UTF-8" ?>
      <autor>
            <nombre>Rodrigo</nombre>
            <apellidos>
                  <apellido>Argüello</apellido>
            </apellidos>
            <apellidos>
                  <apellido>Flores</apellido>
            </apellidos>
      </autor>
```

Capítulo 3

Requisitos

En este capítulo se describirán los requisitos que deberá satisfacer el sistema de telecontrol del humedal, divididos en requisitos funcionales y no funcionales. Se procederá a definir los requisitos de forma separada, primero para el sistema global y después para la aplicación web. También se explicarán algunas restricciones definidas para cada uno de los elementos.

3.1 REQUISITOS GLOBALES DEL SISTEMA

Los requisitos que se describen a continuación engloban todos los requerimientos y restricciones definidos para el sistema completo, compuesto por la red de sensores inalámbricos y la aplicación web. Se definirán primero los requisitos funcionales y a continuación los no funcionales.

3.1.1 REQUISITOS FUNCIONALES

- **RF01:** La red de sensores deberá muestrear y proporcionar los siguientes parámetros del humedal:
  - Tensión de referencia-working de los potenciostatos 1, 2 y 3
  - Medida de corriente en los potenciostatos 1, 2 y 3
  - Tensión de contra-working de los potenciostatos 1, 2 y 3
  - Medida de potencia en los potenciostatos 1, 2 y 3
  - Temperatura del agua
  - Conductividad del agua

  - pH del agua
  - Temperatura del aire
  - Caudal del agua entrante
  - Ciclo de trabajo de la bomba de entrada
  - Estado de la válvula de entrada
  - Ciclo de trabajo de la bomba de recirculación
  - Precipitaciones
  - Humedad relativa atmosférica
- **RF02:** Se deberán muestrear los datos mencionados en el requisito anterior cada 30 segundos.
- **RF03:** Las motas asociadas a las medidas de la fuente de tensión del humedal, así como las asociadas a las bombas de entrada y recirculación, deben ser también capaces de modificar parámetros en las mismas.
- **RF04:** El nodo central de la red de sensores debe ser capaz de enviar la información obtenida a la aplicación web.
- **RF05:** El sistema debe ser capaz de ser modificado por los operarios de manera remota (vía Internet).
- **RF06:** La monitorización de los parámetros del sistema debe poder ser realizada de forma remota (vía Internet).

3.1.2 REQUISITOS NO FUNCIONALES

- **RNF01:** La red de sensores debe ser inalámbrica.
- **RNF02:** El consumo de los sensores debe ser mínimo.
- **RNF03:** La red de sensores estará suministrada energéticamente únicamente por la propia energía generada por el humedal.
- **RNF04:** La comunicación entre la red de sensores y la aplicación web debe ser vía sockets TCP.
- **RNF05:** No debe existir pérdida de mensajes en la comunicación entre la red de sensores y la aplicación web.
- **RNF06:** La comunicación entre la red de sensores y la aplicación web debe tener autenticación.

3.2 REQUISITOS DE LA APLICACIÓN WEB

Los requisitos que se describen a continuación definen el comportamiento que debe seguir la aplicación web del sistema de telecontrol, que será la interfaz encargada de comunicar al operario con el sistema de control del humedal (situado en la red de

sensores). Como se ha hecho en el apartado anterior, se dividirán en requisitos funcionales y no funcionales.

3.2.1 REQUISITOS FUNCIONALES

- **RF01:** Deben existir 3 roles en la página: Usuario normal, Usuario avanzado y Administrador.
- **RF02:** Los usuarios no registrados sólo pueden tener acceso a las páginas de *login* de la web y del sitio de administración.
- **RF03:** Sólo debe poder registrar usuarios en la web un administrador.
- Cada uno de los roles tendrá diferentes permisos dentro de la web:
- **RF04:** El Usuario normal sólo podrá acceder a la página de monitorización, evolución del sistema e históricos.
- **RF05:** El Usuario avanzado heredará los permisos del Usuario normal y, además, podrá acceder al cuadro de control y enviar cambios al sistema de control.
- **RF06:** El administrador tendrá todos los permisos que se han descrito para los roles anteriores y además podrá acceder a la página de administración de la web.
- La página de monitorización (la página a la que se accede después del login) debe cumplir los siguientes requisitos funcionales:
  - **RF07:** Ser visible sólo para usuarios registrados.
  - **RF08:** Permitir visualizar los últimos parámetros que han llegado del sistema.
  - **RF09:** Los datos que llegan al sistema deben ser mostrados en tiempo real sin necesidad de tener que recargar la página.
- Las especificaciones funcionales de la página del cuadro de control son las siguientes:
  - **RF10:** Debe ser visible sólo para usuarios registrados y como mínimo con el rol de Usuario avanzado.
  - **RF11:** Debe mostrar formulario HTML que permita enviar las modificaciones al sistema de control.
  - **RF12:** Los valores iniciales de los parámetros del cuadro de control deben ser los últimos cambios que se enviaron al sistema (sin importar el usuario que los hizo).
  - **RF13:** Si el valor del parámetro Modo de operación equivale a automático, la página no debe permitir modificar el resto de parámetros sin modificar el Modo de operación previamente.

- **RF14:** Si la conexión con el servidor al que se envían las modificaciones (nodo central de la WSN) no ha podido establecerse, se debe notificar al usuario y no permitirle realizar cambios.

- Las especificaciones funcionales de la página de evolución del sistema son las siguientes:
  - **RF15:** Debe ser visible sólo para usuarios registrados.
  - **RF16:** Debe mostrar de forma clara la evolución temporal de las variables Tensión de referencia y Corriente del sistema.
  - **RF17:** El tipo de representación debe ser una gráfica de puntos.
  - **RF18:** La variable a representar será la media por día de esas variables.

3.2.2 REQUISITOS NO FUNCIONALES

- **RNF01:** La aplicación web debe ser completamente privada (no debe permitir el registro de usuarios si no es a través de un administrador).
- **RNF02:** La aplicación debe tener un mínimo de seguridad añadida aparte de la autenticación.
- **RNF03:** La web no debe contener logos ni imágenes que identifiquen a la organización.
- **RNF04:** La web debe ser lo más simple posible y todo debe estar a mano del usuario.
- Dentro de la página de monitorización:
  - **RNF05:** Los parámetros que representan valores booleanos (error o no error, abierto o cerrado, etc.) deben tener una característica llamativa visualmente (color verde y rojo, por ejemplo) para que el usuario pueda detectar comportamientos anómalos en el sistema de un simple vistazo.
  - **RNF06:** Los datos deben estar organizados en columnas y deben poder observarse todos los datos sin necesidad de bajar en la barra del navegador en un monitor con una resolución mínima de 1280x1024 píxeles.
- **RNF07:** Toda la administración de la web (registrar nuevos usuarios, añadir usuarios a grupos, crear grupos, borrar usuarios, etc.) debe ser de forma gráfica, ya que el administrador no será una persona con conocimientos técnicos.

Capítulo 4

Arquitectura

En este capítulo se hablará sobre la arquitectura del sistema. Primero se describirá la arquitectura que seguirá el sistema completo en el que se engloba el proyecto. A continuación, se procederá a describir de forma más detallada la arquitectura interna de cada una de las partes que conforman el sistema.

4.1 ARQUITECTURA GENERAL DEL SISTEMA

El sistema construido con objeto de la monitorización y telecontrol del humedal inteligente sigue una arquitectura básica cliente-servidor, con una interfaz intermedia entre el sistema de control (la red de sensores que monitoriza el humedal, equipada con el sistema de actuación correspondiente) y el usuario. Esta interfaz es el sistema de telecontrol, el elemento perteneciente al sistema global desarrollado para este proyecto y donde se centra este documento.

El sistema está conformado por varios elementos que interactúan directamente entre sí. A continuación se procederá a describir la función que realiza cada uno en el sistema (de manera global), y se aportará un esquema gráfico para mostrar la interacción entre los mismos.

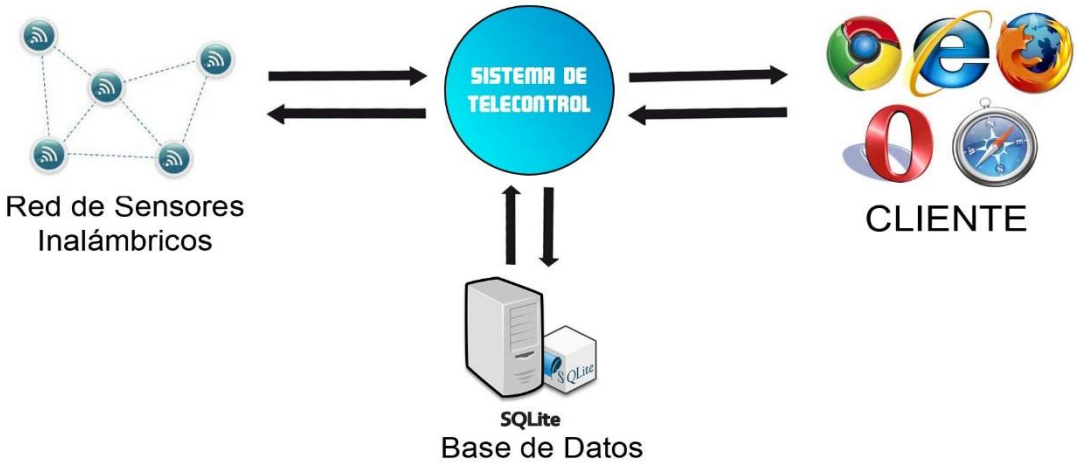

Ilustración 15: Esquema de la arquitectura global del sistema.

- **Red de sensores inalámbrica:** es el elemento encargado de obtener las magnitudes físicas y químicas necesarias para la monitorización del humedal. También está equipado con el sistema de actuación (fuente de tensión, control de válvulas, etc.). Se comunicará con el sistema de telecontrol (aplicación web) a través de sockets TCP mediante el intercambio de mensajes en formato hexadecimal codificado en ASCII.
- **Sistema de telecontrol:** es la interfaz encargada de comunicar al usuario con el humedal. Permite al usuario realizar cambios en el humedal y monitorizar los datos medidos por la red de sensores en tiempo real.
- **Base de datos:** es el sistema de almacenamiento que contendrá toda la información relativa al sistema. Estará gestionada por la aplicación web, y almacenará los datos recibidos con información relativa al humedal, usuarios en la aplicación, cambios enviados al sistema, etc.
- **Cliente:** estará representado por los operarios del sistema a través de un cliente HTTP (navegador web). Será el personal que se encargue del funcionamiento del humedal, revisión de parámetros de calidad del agua y depuración, etc.

4.2 ARQUITECTURA DE LA RED DE SENSORES INALÁMBRICOS

La red de sensores inalámbricos será el subsistema encargado de muestrear ciertos parámetros de interés para la depuración del agua. Cabe destacar que el diseño de

esta arquitectura no ha sido objetivo del TFG desarrollado, pero se considera de gran importancia la descripción de la misma para la comprensión del resto de las arquitecturas descritas.

Como en todas las WSN, existirá un nodo central que coordinará al resto, siendo el encargado de intercomunicar al resto de los nodos con la aplicación web y viceversa.

Además, los nodos complementarios, serán nodos capaces de realizar cambios físicos en el humedal, según las órdenes recibidas por el nodo central. El nodo central ejecutará un algoritmo de control encargado de controlar el humedal según la información recibida por el sistema de telecontrol.

La red de sensores sigue una topología en forma de estrella, tal y como se muestra en la figura X.

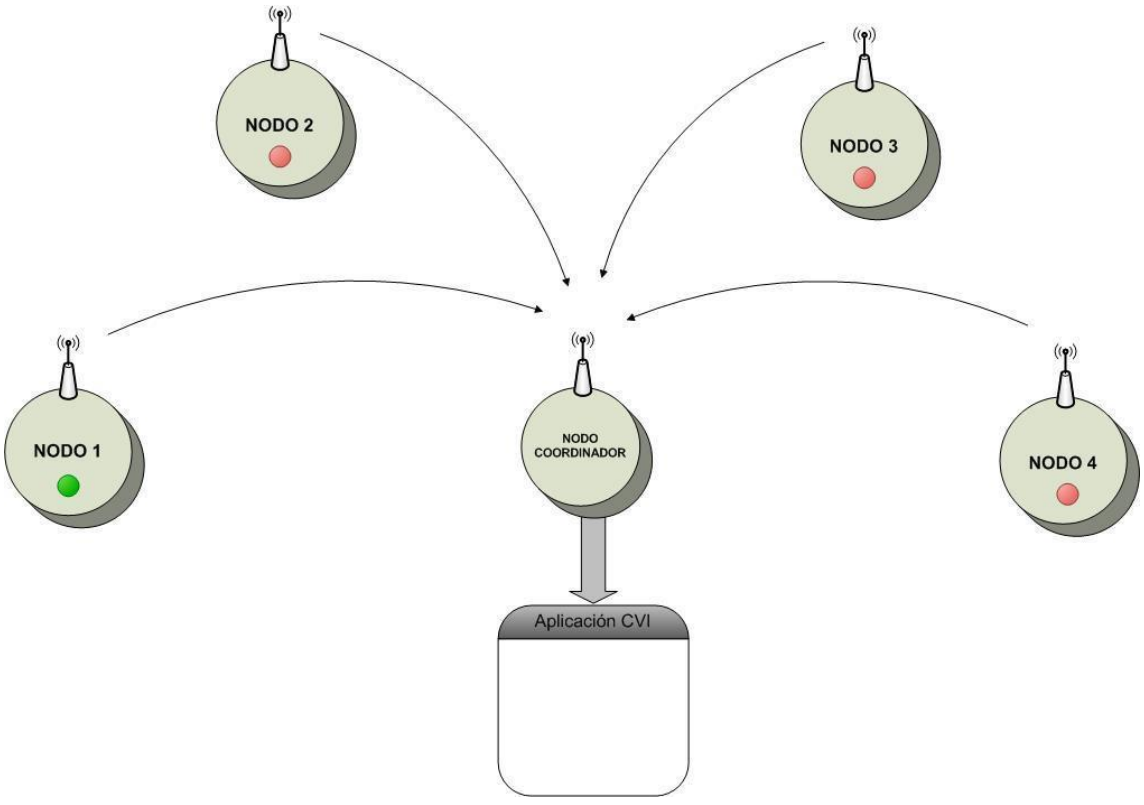

Ilustración 16: Arquitectura de la red de sensores inalámbricos. Fuente: Memoria técnica oficial del proyecto Smart Wetland.

La red de sensores estará conformada por los siguientes tipos de nodo/mota:

- **Nodo central:** en la figura se corresponde con el nodo coordinador. Es el encargado de recibir datos de todos los nodos de la red, además de

comunicarse directamente el sistema de telecontrol. También es el nodo encargado de ejecutar las órdenes en el sistema comunicándose con los nodos de control. Este nodo no tendrá limitaciones de batería ni de capacidad de cómputo, ya que será un PC industrial situado en una zona asegurada del humedal.

- **Nodo de control:** en la figura se corresponde con el nodo 1. Es el nodo que proporciona información de la fuente de tensión (potencia, voltaje, corriente aplicada, información de sobre-tensión, etc.), además de controlar esta misma fuente de tensión (lo que es lo mismo, controlar el humedal). El nodo principal le manda información de la tensión que debe poner entre el ánodo y la referencia y realiza los cambios a la vez que le envía información durante el proceso.
- **Nodos sensores tipo 1:** en la figura se corresponden con 2 de los nodos 2, 3 y 4. Estos nodos son los encargados de medir características del humedal y ciertos parámetros que indican la calidad de la depuración del agua.
- **Nodo sensor tipo 2:** en la figura se corresponde con 1 de los nodos 2, 3 y 4. Este nodo es un voltímetro (informa de la tensión que hay entre otros dos puntos determinados del humedal) y un medidor de temperatura de las aguas.

4.3 ARQUITECTURA DEL SISTEMA DE TELECONTROL

El elemento presentado como Sistema de telecontrol al describir la arquitectura general de la aplicación, tal y como se ha explicado anteriormente, es el encargado de comunicar las acciones del usuario con el sistema de control situado en la red de sensores, además de permitirle visualizar datos/gráficas en tiempo real. Esta arquitectura ha sido diseñada de forma completa durante las primeras fases del desarrollo de este TFG.

La arquitectura interna que sigue es la clásica arquitectura cliente-servidor. Sin embargo, en este subsistema existirán varios clientes (el usuario y la red de sensores, cuando envíe datos provenientes del humedal), y también se comportará como cliente de cara a la red de sensores cuando el usuario realice cambios en el sistema.

El elemento principal que forma este subsistema es la aplicación web desarrollada escrita en Django. Sin embargo, existen varios elementos intermedios, aparte de la aplicación web, con el objetivo de prestar la funcionalidad deseada en el sistema.

A continuación se muestra de forma gráfica como se produce la intercomunicación dentro de la aplicación, y se explicarán la funcionalidad específica de cada uno de los elementos.

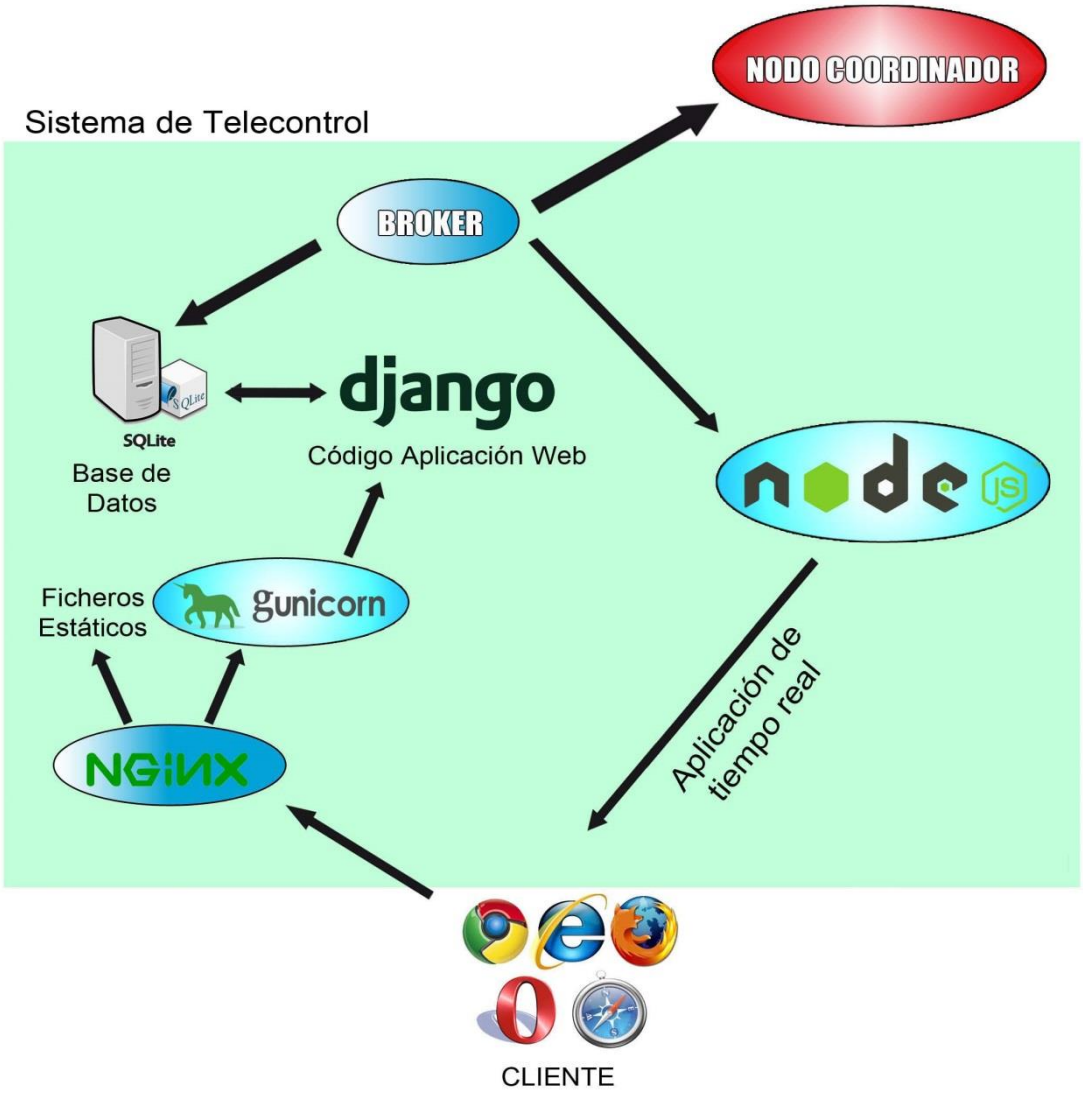

Ilustración 17: Esquema gráfico del sistema de telecontrol.

- **Servidor intermediario (broker):** es un servidor intermedio situado entre la aplicación web y el nodo central de la red de sensores. Realiza las siguientes funciones:
  - Escucha de forma indefinida en un socket TCP en un puerto libre acordado. Espera tramas procedentes de la red de sensores, para procesarlas tras haber hecho las comprobaciones pertinentes en la trama.

- Es el encargado de guardar las tramas con formato hexadecimal formato hexadecimal codificado en ASCII recibidas, para transformarlas al tipo de datos que maneja la aplicación web, asignando fecha y hora.
- Por último, otra función que realiza es la de enviar los datos al servidor Node, el elemento del sistema de telecontrol encargado de aportar la funcionalidad de la visualización de datos en tiempo real en la web, en formato JSON.

- **Servidor Node:** es el encargado de conseguir la actualización de datos en tiempo real en la página de monitorización. El cliente Node está situado dentro de la aplicación web, dentro de la plantilla HTML de monitorización.
- **Aplicación web (Django):** en ella se situará toda la funcionalidad orientada al usuario. Realizará consultas en la base de datos (para la lectura de datos provenientes de la red de sensores y la gestión de claves y sesiones de usuario), además de realizar escrituras a través del sitio de administración y al registrar cambios del cuadro de control.
- **Servidor web:** es el encargado de proveer de acceso a la aplicación web a los usuarios con acceso a internet. Atiende peticiones HTTP de los clientes y les proporciona el contenido solicitado de la web.
- **Cliente:** en este caso el cliente es el mismo que el descrito en la sección de la arquitectura global del sistema.

4.3.1 ARQUITECTURA DE LA APLICACIÓN WEB

En esta sección se describirá el proceso interno de la aplicación Django, la función que cumplen los elementos que la forman y la interacción entre los mismos.

Como se describió en el capítulo 2 al hablar sobre Django, las aplicaciones escritas con este *framework* siguen el patrón/arquitectura de software MTV. Sin embargo, además de estas 3 principales entidades, existen otros elementos que interactúan cada vez que se recibe una petición HTTP. A continuación se enumeran y se describe su función específica:

- **Django middleware:** es el encargado de realizar ciertos procesamientos previos. Django posee una gran cantidad de herramientas de middleware, y el desarrollador puede activar y desactivar las que considere necesarias para su aplicación. En esta aplicación, la arquitectura de middleware presente es la siguiente:

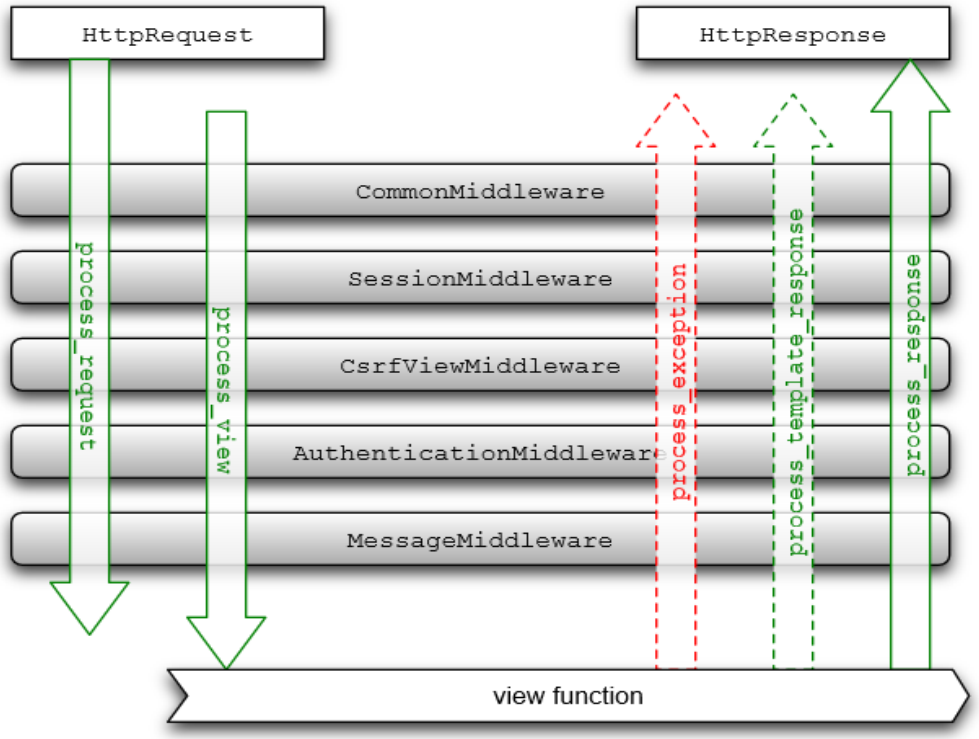

Ilustración 18: Arquitectura del middleware de Django. Fuente: docs.djangoproject.com.

La función de cada uno de los elementos del middleware es la siguiente:

- **CommonMiddleware:** aporta funcionalidades extra en el ámbito de la aplicación, tales como el negar el acceso a ciertos agentes de usuario, normaliza URLs, etc.
- **SessionMiddleware:** almacena y gestiona sesiones de usuario de forma automática.
- **CsrfViewMiddleware:** protege contra ataques Cross-site scripting (explicado en la sección de seguridad de Django en el capítulo 2). Genera cookies para la protección contra este tipo de ataques y automatiza la comprobación de las mismas al recibir cualquier petición HTTP dentro de la aplicación.
- **AuthenticationMiddleware:** middleware que realiza la función de autenticación de usuarios en la página de forma automática.
- **MessageMiddleware:** middleware de Django utilizado para el envío bidireccional de mensajes dentro de la aplicación, con información sobre la sesión, cookies, etc.

- **Django URL dispatcher:** es el encargado de mapear la URL solicitada con la vista que debe ser ejecutada.
- **La vista:** es el código que se ejecuta cuando se recibe una petición HTTP a través de una URL en concreto. En la aplicación, dependiendo de la vista, realizan funciones diferentes, pero todas realizan operaciones con la base de datos, además de devolver respuestas HTTP al cliente.
- **El modelo:** es el encargado de definir las estructuras de datos de la aplicación, así como de proveer de la conexión entre la aplicación y la base de datos mediante el mapeador objeto-relacional, que relaciona las clases Python que definen los datos con la base de datos, dando igual la tecnología utilizada para la misma. Define estructuras de datos tales como los datos recibidos del humedal, los usuarios, etc.
- **La plantilla:** es la parte de la aplicación encargada de la presentación visual (interfaz gráfica). Presenta la información al usuario mediante plantillas HTML, y lo hace de forma dinámica mediante la herramienta de Django template context processor, de la que se habló en el capítulo 2. En la aplicación, es la encargada de presentar los datos del humedal, gráficos, el formulario para realizar cambios en el sistema y en definitiva, es la manera en la que el usuario interactúa con la aplicación a través de un navegador web.

4.3.2 ARQUITECTURA DEL SERVIDOR WEB

Como se ha mencionado anteriormente, el servidor web es el encargado de "conectar" la aplicación web con internet, con la posibilidad de que los usuarios accedan a ella a través de internet.

La arquitectura interna que sigue el servidor web elegido para el proyecto es la siguiente:

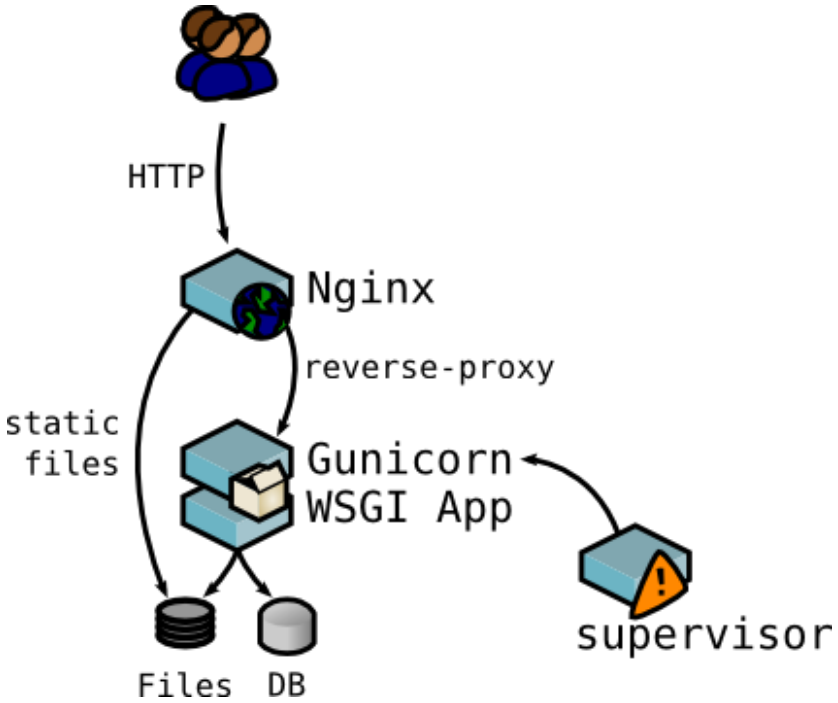

Ilustración 19: Configuración de Nginx junto con Gunicorn como servidor web. Fuente: www.deltalima.net

Nginx actúa como servidor web principal. Sin embargo, sólo se encarga de servir el contenido estático: JavaScript, hojas de estilo CSS, fuentes, imágenes, etc.

Las peticiones del cliente que solicitan el contenido no estático de la web (plantillas dinámicas HTML, código Python, etc.) son redirigidas por Nginx al puerto en el que está Gunicorn, para que este se encargue de actuar como servidor web sólo para los ficheros mencionados.

Esta arquitectura resulta óptima en cuanto al consumo de recursos de hardware debido a las características de ambos servidores web.

Capítulo 5

Diseño

En este capítulo se hablará sobre las principales medidas de diseño del proyecto.

Se explicará principalmente el diseño del sistema de telecontrol (conformado, a su vez, por diferentes elementos, tal y como se explicaba en el capítulo 4), ya que es el elemento desarrollado en este trabajo, a su vez enmarcado dentro de un proyecto global, tal y como se ha explicado en capítulos anteriores.

Sin embargo, también se explicarán algunos aspectos que no pertenecen únicamente al sistema de telecontrol, pero tienen que ver directamente con él, tal y como es la estructura de comunicación entre la red de sensores y la aplicación web.

5.1 DISEÑO DEL MODELO DE DATOS

En esta sección se describirá el modelo de datos manejado en el sistema de control.

Como en Django primero se define el modelo de datos en la aplicación y luego él crea las tablas de manera automática, se procederá a describirlo en este orden.

5.1.1 ESTRUCTURA DE LOS DATOS EN LA APLICACIÓN

Los siguientes modelos de datos son los manejados dentro de la aplicación web. Algunos de ellos ya venían implementados por Django y otros han sido diseñados específicamente para esta aplicación.

- **User:** Es la representación de los usuarios en la aplicación. Esta estructura viene implementada de forma previa en Django. Contiene los siguientes atributos, siendo algunos de ellos opcionales:
  - **username:** atributo obligatorio. Es una cadena de caracteres alfanuméricos que representan el nombre del usuario.
  - **first_name:** atributo opcional. Cadena de caracteres alfanuméricos que representan el nombre de pila del usuario.
  - **last_name:** atributo opcional. Cadena de caracteres alfanuméricos que representa el apellido del usuario.
  - **email:** cadena de caracteres que representa la dirección de email del usuario.
  - **password:** es el campo dónde se almacena información sobre la contraseña, no la contraseña en sí.
  - **groups:** especifica los grupos a los que pertenece el usuario.
  - **user_permissions:** especifica los permisos del usuario en la página.
  - **is_staff:** atributo *booleano* que designa si el usuario puede acceder al sitio de administración.
  - **is_active:** atributo *booleano* que informa si el usuario no tiene actividad en la página desde un tiempo considerable.
  - **is_superuser:** atributo *booleano* que especifica si el usuario tiene todos los permisos sin especificarlo explícitamente.
  - **last_login:** informa respecto a la fecha en la que el usuario inició sesión la última vez.
  - **date_joined:** atributo que informa sobre la fecha en la que se creó la cuenta del usuario.
- **RealtimeData**: es la estructura de datos que contiene los valores de los diferentes parámetros del humedal. Esta estructura ha sido definida durante la etapa de diseño de este TFG. Todos ellos salvo el campo created_at, que es un campo de tipo *DateTimeField*, son campos de tipo *FloatField* de Django. No se enumeran todos, ya que en secciones posteriores se describirá el modelo de comunicación sentido servidor-cliente, y los datos aquí definidos representan lo mismo.
- **ClientPacket**: es la estructura de datos definida dentro de la aplicación para manejar los cambios que se envían desde la misma hacia el servidor de la WSN. Al igual que la estructura anterior, ha sido definida durante la etapa de diseño de este TFG. Todos los campos que contiene definen lo mismo que lo que se describirá en el apartado posterior que describa la comunicación sentido cliente-servidor. Sin embargo, se añaden campos

tales como la fecha y el usuario que envió los cambios al servidor. Los tipos de datos utilizados en esta estructura de datos son *DecimalRangeField*, para números decimales que deben estar en un rango determinado; *BooleanField*, *IntegerField* y *DecimalField* para campos de tipo *boolean*, *int* y decimales respectivamente.

5.1.2 DISEÑO DE LA BASE DE DATOS

La base de datos es creada automáticamente por Django una vez los modelos de la aplicación han sido creados y la tecnología para la base de datos es elegida (Django crea las tablas, consultas SQL, etc. para adaptarse a los modelos de la aplicación, dependiendo de la tecnología escogida entre las que Django soporta) por tanto no es necesario en este caso explicar de forma explícita el modelo de datos en la base de datos.

Por tanto, la opción de diseño en este caso ha sido la elección de la tecnología a utilizar en la base de datos.

Una de las restricciones existentes en el proyecto es que la tecnología para la base de datos debía no ser de pago incluso con fines comerciales. Por tanto, se barajan las siguientes opciones:

- SQLite, cuyas ventajas y desventajas son las siguientes:
  - Ventajas: es que es un sistema de gestión de bases de datos relacional muy ligero en cuanto a consumo de recursos y fácil de poner en funcionamiento (es la que viene configurada por defecto en Django). Además es fácilmente portable debido a que la base de datos está en un solo fichero en el disco.
  - Desventajas: complicado para acceder de manera remota (se necesita montar el sistema de ficheros o utilizar software adicional) y no permite concurrencia (toda la base de datos es bloqueada al hacer alguna consulta).
- PostgreSQL, cuyas ventajas y desventajas son las siguientes:
  - Ventajas: es más potente que SQLite, permitiendo concurrencia, conexión remota (sigue el modelo cliente-servidor, por lo que es un proceso que escucha en un puerto concreto) y proporciona integridad de datos.

- Desventajas: a pesar de ser un sistema de gestión de bases de datos muy potente, es mucho más "pesado" que SQLite en cuanto a consumo de recursos de hardware.

Principalmente, debido a las limitaciones de hardware que puede haber en el equipo situado en el humedal, a que la estructura de datos de la aplicación será relativamente simple y sólo una de las partes (la aplicación) se encargará de hacer consultas con la base de datos, se ha escogido SQLite como sistema de gestión de bases de datos para el proyecto.

5.2 DISEÑO DE LA ESTRUCTURA DE COMUNICACIÓN

En este sección se describe la estructura de comunicación diseñada para el intercambio de mensajes entre la red de sensores y la aplicación web. Esta estructura de comunicación estaba previamente diseñada y no ha sido objetivo de este TFG. Sin embargo, la explicación de la misma es de vital importancia para la comprensión del presente documento.

Los mensajes intercambiados entre estas 2 partes contendrán variables codificadas en hexadecimal. Según el tipo de variable, se codificarán de forma diferente, según el formato descrito a continuación:

- Los datos de tipo *int* se codificarán en 32 bits, lo que equivale a 8 caracteres hexadecimales.
- Los datos de tipo *double* (números decimales de doble precisión) se codificarán en 64 bits, lo que equivale a 16 caracteres hexadecimales.
- Los datos de tipo *boolean* se codificarán en 8 bits, lo que equivale a 2 caracteres hexadecimales.

La explicación del diseño de comunicación se subdividirá en 2 partes: la estructura de comunicación diseñada para los mensajes procedentes del cuadro de control de la aplicación web (cliente para este caso) dirigidos al sistema de control del humedal (servidor) y la estructura de los mensajes procedentes de la red de sensores (servidor) hacia la aplicación web (cliente).

5.2.1 COMUNICACIÓN SENTIDO SERVIDOR-CLIENTE

Los mensajes que van en este sentido de la comunicación contienen información recogida por los nodos sensores del humedal, además de otros campos adicionales.

Estos mensajes contienen un campo de tipo *int* y 34 campos de tipo *double*, lo que equivale a un total de 276 Bytes por mensaje.

En la siguiente tabla se describen cada uno de los campos de los mensajes enviados por el servidor (excepto el primer campo de tipo *int*):

Campo	Nombre variable	Tipo	Rango valores [Unidades]	Definición
0	Vref_1	*Double*	[-1,1]	Medida tensión referencia-working potenciostato.
1	I_1	*Double*	[-1,1]	Medida corriente potenciostato.
2	Vcw_1	*Double*	[-8,8]	Medida tensión contra-working potenciostato.
3	Potencia_1	*Double*	[-8,8]	Medida potencia entregada.
4	Vcw_1 in range	*Double*	0, 1	Mide si está o no la tensión contra-working dentro de un rango determinado.
5	I_1 in range	*Double*	0, 1	Mide si la corriente medida está dentro de un rango determinado.
6	Error de Comun. Con tarjeta_1	*Double*	0, 1	Se ha producido un error de comunicación con la tarjeta.
7-8-9-10-11-12-13	…	…	…	Lo mismo para el potenciostato 2 (calidad del agua).
14-15-16-17-18-19-20	…	…	…	Lo mismo para el potenciostato 3 (calidad del agua).
21	Vref_1-Vref_2	*Double*	Sin definir	Diferencia de potencial entre el electrodo de referencia 1 y el electrodo de referencia 2.
22	Temperatura Agua	*Double*	Sin definir	Temperatura del agua.
23	Vref_1-Vref_2 in range	*Double*	0, 1	Mide si la tensión está dentro de un rango determinado.
24	Error de Comun. Con tarjeta 4	*Double*	0, 1	Se ha producido un error en la comunicación con la tarjeta.
25	Conductividad	*Double*	Sin definir	Conductividad de las aguas.

26	pH del agua	*Double*		Medidor de pH.
27	Temperatura Aire	*Double*	[-10,50]	Medida de la temperatura del aire.
28	Caudal de agua	*Double*	Sin definir	Caudal de agua entrante.
29	Bomba de entrada	*Double*	Sin definir	Ciclo de trabajo de la bomba de entrada por hora.
30	Válvula de entrada	*Double*	0, 1	Estado de la válvula 0 - Cerrada 1 - Abierta
31	Bomba de recirculación	*Double*	Sin definir	Ciclo de trabajo de la bomba de recirculación por hora.
32	Lluvia	*Double*	Sin definir	Precipitaciones.
33	Humedad	*Double*	[0,100]	Humedad relativa atmosférica.

Tabla 1: Campos de un mensaje en la comunicación sentido Servidor-Cliente

Además, se proponen los siguientes cambios en la futura estructura de comunicación para este sentido:

- De cara a la optimización, cambios en el tipo de los campos (muchos de ellos podrían no ser necesariamente de tipo double), para el ahorro de Bytes en la comunicación.
- Inclusión de un campo de autenticación con una contraseña específica para la comunicación entre estas 2 entidades.

5.2.2 COMUNICACIÓN SENTIDO CLIENTE-SERVIDOR

Los mensajes que van en este sentido de la comunicación contienen información que define los cambios que desea hacer un operario en el sistema de control del humedal.

Estos mensajes contienen 2 campos de tipo *int*, 5 campos de tipo *double* y 4 campos de tipo *boolean*, lo que equivale a un total de 52 Bytes por mensaje.

En la siguiente tabla se describen cada uno de los campos de los mensajes enviados por el servidor (al igual que en la tabla anterior, no se muestra el primer campo de tipo *int* que especifica el tamaño de la trama):

Campo	Nombre variable	Tipo	Rango valores [Unidades]	Definición
0	Vref_1	*Double*	[-1,1]	Tensión de referencia de electrodo 1.
1	Electrode Activation 1	*Boolean*	[0,1]	Permite activar o desactivar el potenciostato 1.
2-3	…	…	…	Lo mismo para el potenciostato 2 (calidad del agua).
4-5	…	…	…	Lo mismo para el potenciostato 3 (calidad del agua).
6	Modo_op	*Integer*	0,1,2	Modo de operación 0 - Automático 1 - Enriquecimiento 2 - Manual
7	Bomba de entrada	*Double*	Sin definir	Ciclo de trabajo de la bomba de entrada por hora.
8	Válvula de entrada	*Boolean*	True, False	Estado de la válvula False - Cerrada True - Abierta
9	Bomba de recirculación	*Double*	Sin definir	Ciclo de trabajo de la bomba de recirculación por hora.

Tabla 2: Campos de un mensaje en la comunicación sentido Cliente-Servidor

Además, se proponen los siguientes cambios en la futura estructura de comunicación en este sentido:

- Inclusión de un campo denominado no modificar. Este campo tendría como objetivo informar al sistema de control del humedal de los campos con los que no debe hacer nada, simplemente dejar como estar. Debido a que en el mensaje hay 10 campos relativos a los cambios, este campo sería un número binario de 10 bits, en el que un 0 en la posición 0 significa que debe hacer caso omiso al valor del campo Vref_1, un 1 en la posición 2 significa que debe modificar la variable Vref_2 con el valor contenido en el campo Vref_2, y así sucesivamente.
- Al igual que en el apartado anterior, se sugiere la inclusión de un campo de autenticación con una contraseña específica para la comunicación entre estas 2 entidades.

5.3 DISEÑO DE LOS ALGORITMOS DE CONTROL

5.3.1 ALGORITMO SERVIDOR BROKER

El servidor *Broker*, tal y como se describió en el capítulo 4, es un intermediario entre la red de sensores y la aplicación web.

Es un servidor TCP/IP implementado con sockets TCP de la librería SocketServer de Python.

El algoritmo de control diseñado para este servidor es el siguiente:

1. Abre un socket TCP en un puerto libre definido previamente (50007) y se dispone a escuchar conexiones indefinidamente.
2. Cuando recibe un mensaje, espera hasta que el mismo tenga al menos una longitud de 8 (el número de caracteres hexadecimales correspondientes al *int* del *checksum*).
3. Cuando lee este *checksum*, continúa almacenando en el buffer el número de caracteres indicados en el campo *checksum*.
4. Cuando hay un mensaje completo (34 campos de tipo *double*, que en longitud hexadecimal se corresponde con 544 caracteres), se procesa, traduciendo los datos hexadecimales al valor correspondiente.
5. Cada vez que se procesa un mensaje, se almacena en la base de datos y es enviado al servidor Node en formato JSON.

5.3.2 ALGORITMO SERVIDOR NODE

El servidor Node, tal y como se explicó en el capítulo 4, es el encargado de hacer que los datos se actualicen en la página web en tiempo real.

El algoritmo diseñado para este servidor es el siguiente:

1. El servidor Node espera de forma indefinida en su bucle de eventos mensajes de un cliente en concreto en el puerto 4000 de la máquina servidora.
2. Cuando llega un mensaje, se recoge y se reenvía mediante socket.io hacia la URL /data de la web, dónde está situado el código Javascript del cliente Node.

3. El cliente Node, que previamente se ha conectado al servidor Node, recoge los mensajes una vez le llegan, analiza el mensaje, lo convierte al formato JSON y procede a colocarlo en cada elemento HTML de la página.

5.4 DISEÑO DE LA APLICACIÓN WEB

En este apartado se hablara sobre las principales decisiones de diseño relativas al sistema desarrollado e implementado, que es la aplicación web. Primero se hablará de la elección del lenguaje/*framework* utilizado para desarrollar la aplicación y después se hablará del diseño a la hora de implementar la aplicación.

5.4.1 ELECCIÓN DEL LENGUAJE/FRAMEWORK PARA LA APLICACIÓN

Tal y como se habló en el capítulo 2 de este documento, existen multitud de lenguajes de programación para el desarrollo de aplicaciones web del lado del servidor.

En el capítulo 2 se analizaron multitud de ventajas que ofrece el lenguaje de programación Python, así como las ventajas que ofrece en concreto el *framework* Django.

Las principales características que han hecho que Python/Django sea el lenguaje escogido para el desarrollo del proyecto son las siguientes:

- La seguridad ofrecida por defecto por Django ha sido uno de los principales puntos, debido a los requerimientos de privacidad del proyecto.
- Todos los atajos/funciones que ofrece Django para agilizar procesos típicos a la hora de desarrollar aplicaciones web.
- Amplia cantidad de tutoriales, gran comunidad de desarrolladores para la resolución de dudas y problemas, etc.

Además, al no haber restricciones de integración de la página con otros servicios (se ha desarrollado un sistema desde cero), se ha elegido el lenguaje que se ha considerado óptimo para el desarrollo del proyecto.

5.4.2 DISEÑO DE LA APLICACIÓN WEB

El proyecto Django que implementa toda la funcionalidad de la aplicación web se ha dividido a su vez en 3 módulos. Cada uno de ellos sigue la arquitectura MTV, sobre la que se habló en el capítulo 4 de este documento e implementan las diferentes funcionalidades, especificadas en el capítulo 3, que son las siguientes:

- **cuadro_control**: es el módulo encargado de implementar la funcionalidad de realizar cambios en el sistema a través de la página web. Este módulo está conformado a su vez por:
  - **views.py**: contiene todas las funciones y operaciones que conforman la vista dentro de este módulo. Principalmente se encargan de validar el formulario de datos ofrecido al usuario, transformar estos datos al formato hexadecimal formato hexadecimal codificado en ASCII y enviarlos a través de un *socket* TCP con destino el servidor de la WSN.
  - **models.py**: aquí se define el modelo de datos utilizado en este módulo. Este modelo de datos se ha mencionado anteriormente y define los valores de los parámetros que se desean modificar en el humedal, además del usuario que realizó los cambios.
  - **templates**: en este directorio están almacenados los ficheros HTML que contienen la vista ofrecida al usuario. Además de implementar el formulario que se ofrece al usuario para que lo rellene, se implementan diferentes funciones en Javascript para mejorar la experiencia del usuario en la página, tal y como es avisar si la comunicación con el servidor de la WSN es correcta, informar si los cambios se han realizado correctamente, mostrar al usuario el último valor que se ha ajustado en ese campo, etc.
  - **forms.py**: en este módulo se especifica las restricciones y campos que debe tener el formulario enviado en esta página.
- **monitorizacion**: es el módulo en el que se implementa la visualización de datos del humedal en tiempo real. Como el anterior módulo, se subdivide a su vez según la arquitectura MTV:
  - **views.py**: aquí se implementa la vista. Existen funciones para inicializar los datos visualizados en la página por primera vez, así como funciones de utilidad para darle formato (color, negrita, etc.) a los datos que van llegando en tiempo real a través del servidor Node.
  - **models.py**: aquí se definen todos los campos que se reciben de la WSN para ser manejados por la aplicación.
  - **templates:** en este directorio está almacenado el fichero HTML que contiene la visualización de la página. En este caso importa las funciones Javascript (cliente Node) en el directorio static de este mismo módulo.
  - **static**: aquí se almacenan los ficheros estáticos utilizados en la aplicación, tal y como son imágenes, Javascript y CSS. En este caso es especialmente importante, ya que aquí se sitúa el cliente Node utilizado

en esta página, que recibe datos en tiempo real y los "coloca" dentro de la página.

- **graficos**: es el módulo que contiene la funcionalidad para representar gráficos. Contiene los siguientes elementos:
  - **views.py**: contiene diferentes funciones que seleccionan de la base de datos el conjunto de datos a representar.
  - **templates**: aquí están los ficheros HTML que contienen las diferentes representaciones gráficas. Se especifica el tamaño del cuadro que contiene la gráfica, el lugar en el que está situado, etc.
  - **static**: al igual que en el módulo anterior, aquí está todo el código Javascript que realiza la representación de las gráficas, características de la representación, propiedades de la gráfica, etc.

Además, existen ciertos elementos en el diseño de la aplicación que actúan a nivel global, y son los siguientes:

- **settings.py**: aquí se especifican numerosas opciones para el proyecto Django. La más importante es la configuración de la base de datos, en la que se especifica el nombre, la ruta relativa, etc. pero también se especifican rutas de ficheros estáticos, la clave secreta de la aplicación, etc.
- **static**: aquí se almacenan todos los ficheros estáticos que son globales para toda la aplicación. Aquí se almacena, por ejemplo Bootstrap y JQuery, en css y js respectivamente, imágenes que aparecen en la plantilla principal, etc.
- **templates**: este es el directorio global para las plantillas del proyecto. Aquí se almacena la plantilla base.html en la que está el esqueleto de todas las vistas de la página (el menú superior e izquierdo) y se importan los principales css y js utilizados en todas las plantillas.

Por último, existe un directorio llamado scripts en los que están situados los servidores Node y *Broker,* sobre los que se ha hablado anteriormente en este documento, además de ciertas herramientas creadas con fines de testeo, tales como *scripts* para la generación de clientes y mensajes y otro tipo de utilidades.

5.4.3 DISEÑO DE LA INTERFAZ GRÁFICA

Uno de los requisitos no funcionales para la aplicación es que la interfaz gráfica únicamente fuera simple, en la que se pudiera acceder a todo de un simple vistazo, y que además no contuviera logos identificativos.

Por tanto, para diseñar las hojas de estilo CSS se ha elegido utilizar Twitter Bootstrap, un conjunto de funciones previamente diseñadas para dar estilo al sitio web y no tener que escribir las hojas de estilo CSS desde cero. Esto supone un ahorro de tiempo considerable, y el objetivo conseguido era el requerido, tal y como se muestra en las ilustraciones 21, 22, 23, 24 y 25 de este documento.

5.4.4 DISEÑO DE LOS GRÁFICOS E HISTÓRICOS

Para el diseño de esta parte de la aplicación web se han estudiado diferentes alternativas, cada una con diferentes ventajas e inconvenientes, que son las siguientes:

- Matplotlib y otras librerías de Python para la representación de gráficas: la principal desventaja de esta solución es que, al realizar la representación directamente en el servidor, el gráfico es estático y el usuario no puede interaccionar en él, por tanto esta solución queda descartada.
- Realización de los gráficos en el navegador con Javascript: esta es la solución escogida debido a la capacidad de crear gráficos dinámicos en los que el usuario pueda modificar el periodo de tiempo que desea visualizar, mostrar cuadros de ayuda, posibilidad de ampliar un día simplemente haciendo *click*, etc.

Dentro de esta última alternativa, existe una enorme cantidad de librerías para la realización de gráficos en Javascript. Se han analizado y probado diferentes alternativas, tales como JIT, jGraph, Flot, nvd3 y Highcharts.

Todas ellas ofrecen una funcionalidad similar, sin embargo se ha optado por el uso de Highcharts debido a los siguientes motivos:

- Gran cantidad de ejemplos, buena documentación y enorme variedad de gráficos.
- Facilidad de poner en funcionamiento y uso del formato JSON para establecer el conjunto de datos para representar.
- Ofrece un tipo de gráfico, Highstock, que se adapta exactamente al tipo de gráfico necesitado en el proyecto: capacidad de visualizar en un espacio pequeño la evolución completa del sistema, con barra de navegación, mensajes emergentes de ayuda, posibilidad de establecer el periodo temporal de visualización, etc.

Ilustración 20: Ejemplo de gráfico Highstock de la librería Highcharts

5.4.5 DISEÑO DEL SERVIDOR WEB

Para el diseño del servidor web que atenderá peticiones de los usuarios que soliciten la página, existe una gran cantidad de alternativas que han sido estudiadas antes de llegar a la solución elegida. Sin embargo, la elección ha estado entre 2 de ellas, que son Apache Web Server y Nginx.

Ambos servidores web son libres, de código abierto y multiplataforma. La principal diferencia entre ambos es la arquitectura que siguen ambos. Tal y como se explicó en el capítulo 2 de este documento, Nginx sigue una arquitectura basada en eventos. Apache es un servidor web que sigue una arquitectura basada en procesos. Debido a esto, y especialmente debido a las limitaciones de prestaciones que sufrirá el equipo que deba hacer de servidor web en el humedal, Nginx es una mejor solución ya que este tipo de arquitectura consume menos memoria y recursos de la máquina servidor.

Por tanto, dado que la elección ha sido Nginx, existe una configuración óptima para proyectos Django, que es el uso de Nginx como servidor principal sirviendo únicamente el contenido estático junto con Gunicorn, redirigiendo el contenido no estático a este último, tal y como se ha explicado en el capítulo 4 de este documento.

Capítulo 6

Implementación

En este capítulo se explicarán los puntos más importantes en el proceso de implementación del sistema de telecontrol. Se explicará en más detalle los aspectos que se consideran de mayor importancia, incluyendo fragmentos de código significativos de la aplicación desarrollada.

Es también de importancia mencionar que, a pesar de haber hablado en capítulos anteriores sobre la WSN del sistema, en este capítulo no se explicará el proceso de implementación de la misma debido a que, aunque el sistema de telecontrol interactúa directamente e intercambia mensajes con la misma, no ha sido objetivo de este proyecto.

6.1 IMPLEMENTACIÓN DEL SERVIDOR INTERMEDIARIO

Como se ha explicado en capítulos anteriores, existe un servidor que hace de intermediario entre la aplicación web y la WSN, encargado de escuchar tramas hexadecimales que contienen información sobre el humedal, para después "traducirlas", guardarlas en la base de datos y redirigirlas al servidor Node. En el capítulo 5 se explicó el algoritmo diseñado para este servidor, por lo que aquí se explicará cómo se ha implementado este algoritmo.

Para la implementación del servidor se ha utilizado la librería de Python SocketServer. Esta librería facilita la creación de servidores TCP, de manera que permite atender cada petición en procesos diferentes sin tener que codificarlo de manera explícita (la

librería ya implementa esta funcionalidad). El proceso a la hora de codificar servidores con esta librería es el siguiente:

1. Configurar el comportamiento y parámetros del servidor.
2. Crear el código que se ejecutará con cada petición.

Para la configuración del servidor, sólo se ha ajustado el comportamiento de atender peticiones en procesos diferentes y que la combinación de teclas CTRL+C acabe con todos los procesos. También se han ajustado el *host* y el puerto a *localhost* y al puerto 50007.

Las partes más significativas del código que se ejecuta con cada petición son las siguientes:

```
while(1):
      RCV_BUFFER += self.request.recv(1024)

      if not RCV_BUFFER:
            self.request.close()
            break

      while len(RCV_BUFFER) >= 8:
            characters_to_read = int(RCV_BUFFER[:8], 16)
            RCV_BUFFER = RCV_BUFFER[8:]

            while len(RCV_BUFFER) < characters_to_read:
                  RCV_BUFFER += self.request.recv(1024)

            message = RCV_BUFFER[:characters_to_read]
            RCV_BUFFER = RCV_BUFFER[characters_to_read:]
```

Esta es la primera parte del código que se ejecuta al atender las peticiones. A continuación se explica el código:

- Nada más empezar el bucle (*while*(1)), se escucha mensajes en el medio con un tamaño máximo de 1024 y se guardan en un *buffer* de lectura. El *if* que viene a continuación cierra la conexión en caso de que se reciba una cadena vacía (es la manera en la que se detectan desconexiones con los sockets de Python).

- A continuación se espera a que el mensaje tenga una longitud mínima de 8 caracteres, que es el primer campo del mensaje en el que se indica el tamaño del resto del mensaje, tal y como se explicó en el capítulo 5.
- El siguiente paso es leer el valor de ese campo, pasándolo de formato hexadecimal formato hexadecimal codificado en ASCII a entero, y quitarlo del *buffer* de lectura.
- Recibimos hasta que llenamos un mensaje completo. Una vez realizado esto, se quita el mensaje del *buffer* y procedemos a procesar el mensaje.

Cabe mencionar que este proceso se realiza de este modo debido al modo en que funciona el protocolo TCP. Con este protocolo, se asegura la entrega de la información de forma ordenada, pero no que llegue toda junta (pueden llegar mensajes incompletos, para después llegar el contenido restante o llegar varios mensajes juntos, dependiendo de la congestión de la red). Por tanto esta es la manera en que se ha implementado esta parte.

A continuación se muestra la parte en la que se implementa la traducción de un mensaje hexadecimal al tipo de datos convencional:

```
t_data = data_utils.translate_server_packet(message)
t_data['created_at'] = datetime.datetime.now()
```

Para este proyecto se ha desarrollado un módulo Python llamado data_utils.py en el que se han alojado funciones recurrentes para la transformación y manipulación de datos. A continuación se muestran las partes más importantes de la funcionalidad desarrollada translate_server_packet:

```
def translate_server_packet(serverpacket):

    if len(serverpacket) != 544:
        return False

    data = {}
    data['vref_1'] = hex2float(serverpacket[:16])
    data['i_1'] = hex2float(serverpacket[16:32])
    [...]
```

Dado que todos los campos (excepto el primero, que indica el tamaño del resto del mensaje), son números decimales, se ha implementado una función denominada hex2float para la transformación de un dato hexadecimal que representa un *double* a *float* de Python, la cual se mostrará a continuación (en el fragmento anterior no se muestran todos los campos del diccionario data debido a que todos se transforman de la misma manera):

```
def hex2float(hexstring):

      if len(hexstring) == 16:
            return struct.unpack('!d', hexstring.decode('hex'))[0]
      else:
            return ('Para convertir un string hexadecimal a float
      de doble precision la longitud debe ser exactamente 16
      digitos')
```

El módulo de Python struct es utilizado para la conversión entre valores de Python y estructuras de C representadas por *strings* de Python. Es especialmente útil para manejar datos binarios de ficheros o provenientes de la red, como es en este caso.

Por tanto, lo que realiza esta función es decodificar la cadena hexadecimal para enviársela como parámetro a la función struct.unpack (para pasar de C a Python), además de indicarle con !d que el dato representa un *double.* Como los *double* tienen una longitud de 16 dígitos (64 bits), se comprueba esta condición en el dato de entrada, para que en caso de no coincidir, retornar un error.

El paso final de este servidor *Broker* consiste en guardar los datos en la base de datos (para lo cual se utilizarán funciones de Django que lo hacen automáticamente) y el reenvío de los mismos al servidor Node, sobre el que se ha hablado anteriormente en este documento. Se ilustra esta funcionalidad a continuación:

```
form = RealtimeDataForm(t_data)
if form.is_valid():
      success = form.save()

request_data = data2html(t_data)

r = redis.StrictRedis(host='localhost', port=6379, db=0)
r.publish('event_handler', json.dumps(t_data))
```

En el fragmento de código anterior se realizan varias operaciones, que se explican a continuación:

- El primer paso realizado es validar los datos y transformarlos al modelo RealtimeData utilizado en la aplicación web para manejarlos, para después guardarlo en la base de datos.
- La función data2html realiza ciertos cambios en el diccionario Python para algunos campos que requieren formato HTML (campos que van en rojo o en verde, dependiendo de si son *True* o *False*).
- Finalmente se utiliza el módulo Redis para enviar la información al servidor Node. La primera de las líneas se utiliza para indicar dónde está corriendo el proceso Redis (en la propia máquina, *localhost*, en el puerto 6379). A continuación se envían los datos al canal *event_handler*, uno de los clientes Redis creados dentro del servidor Node. De esta manera, en el servidor Node podríamos crear múltiples clientes Redis para tratar de otro modo o hacer otro tipo de operaciones en los datos, dependiendo del cliente Redis escogido.

6.2 IMPLEMENTACIÓN DEL SERVIDOR Y CLIENTE NODE

Como se ha explicado durante este documento, para conseguir la funcionalidad de la actualización de datos en tiempo real se ha escogido utilizar la tecnología Node.js. En esta sección se describirá como se han implementado el servidor y cliente Node de la aplicación.

6.2.1 IMPLEMENTACIÓN DEL SERVIDOR NODE

En el capítulo 5 se describió el algoritmo definido para este servidor. A continuación se mostrarán fragmentos significativos del código desarrollado junto con la explicación del mismo.

El servidor Node está compuesto principalmente por una etapa de configuración y por las funciones que se ejecutan cuando se detecta un evento determinado.

La etapa de configuración se ejecuta nada más ejecutarse y empezar a correr y realiza configuraciones iniciales en socket.io para que almacene la cookie de autenticación que utiliza Django, además de importar todas las librerías necesarias .También se crea un cliente Redis que se enlaza al canal *event_handler*, tal y como se mencionó en la sección anterior.

A continuación, se muestra el código que se ejecuta cuando se detecta el evento del mensaje a través del cliente Redis (proveniente del *Broker*):

```
var sub = redis.createClient();
sub.subscribe('event_handler');

io.sockets.on('connection', function (socket) {

    //Coge el mensaje de Redis y se lo reenvía al cliente Node
    sub.on('message', function(channel, message){
        socket.send(message);
    });
});
```

La primera función que se muestra, io.sockets.on('connection'), detecta el evento de conexión (redis funciona a través de socket.io). La variable sub está enlazada al canal *event_handler*, por tanto, cuando se reciba un mensaje a través de este canal, se ejecutará la función sub.on('message'), que recoge el mensaje recibido y lo envía a través del websocket (aunque podría ser otro tipo de comunicación, tal y como se explicó en el capítulo Estado del arte).

6.2.2 IMPLEMENTACIÓN DEL CLIENTE NODE

El cliente Node es un cliente muy simple, que cada vez que detecta un evento (recibir un mensaje), realiza conversiones en los datos para mostrarlos en pantalla. A continuación se muestra el código de este cliente:

```
$.getScript("http://" + nodejs_address + ":" + nodejs_port + "/"
+ socket_io_path, function(){

        try {
            var socket = io.connect(nodejs_address, {port:
nodejs_port});
        } catch (err) {
            console.log('Error de conexión')
        }

        socket.on('connect', function(){
            console.log('events.js: connect');
        });

        socket.on('message', function(message) {
            var data = JSON.parse(message)

            for (var field in data){
                var field_id = document.getElementById(field)

                if (field_id!=undefined){
                    field_id.innerHTML = data[field]
                }
            }
        });
    });
```

Previamente a este código mostrado se configuran variables que indican dónde está el servidor Node, además de la ruta a socket.io para poder conectarse al mismo. La función socket.on('message') es la que se ejecuta cuando se recibe un mensaje a través de socket.io. Primero transforma el mensaje a JSON, para después recorrerlo campo por campo. Los id de todos los elementos dentro del documento HTML tienen el mismo nombre que los campos del mensaje recibido. Por tanto, con la función innerHTML ajustamos el valor de ese campo al que contiene el mensaje recibido.

6.3 IMPLEMENTACIÓN DE LA APLICACIÓN WEB

En esta sección se describirá el proceso de implementación de la aplicación web con el *framework* Django. Se procederá a describir la configuración general de la plataforma para después explicar la implementación de cada módulo por separado.

6.3.1 CONFIGURACIÓN GENERAL DEL PROYECTO DJANGO

Para la configuración de la aplicación web en sí misma, se han ajustado varios parámetros y varios módulos Python que tienen tal finalidad. A continuación se explica cada una de los parámetros configurados y cómo se ha llevado a cabo.

6.3.1.1 Configuración de la base de datos

Para configurar la base de datos en Django sólo hay que ir al módulo settings.py del proyecto y añadir un diccionario llamado DATABASES que contenga la información de la/las base/s de dato/s. A continuación se muestra la configuración implementada para el proyecto:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': path.join(BASE_DIR, 'smartwetland.db'),
    }
}
```

Con estas líneas de código se configura la base de datos del proyecto (en este caso sin usuario y contraseña, aunque en caso de necesitarlas, sólo habría que añadir 2 parámetros más al diccionario).

Cuando los modelos de todas las aplicaciones del proyecto Django están creados, sólo hay que añadir estas aplicaciones a la variable INSTALLED_APPS y abrir un terminal y escribir la siguiente línea:

manage.py syncdb

De esta manera, Django crea la base de datos automáticamente, ajustándose a la tecnología, nombre, ruta especificados y a los modelos de todo el proyecto. Además, crea y gestiona las tablas de usuarios, sesiones, cookies, etc. de forma automática.

6.3.1.2 Configuración de las URLs de la página

Tal y como se explicó en el capítulo 2 al hablar sobre el *framework* Django, para configurar las URLs de toda la página web hay que configurar el módulo urls.py del

proyecto Django. A continuación se muestra el fragmento más significativo del módulo urls.py global del proyecto:

```
urlpatterns = patterns('',
    url(r'^grappelli/', include('grappelli.urls')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^login/$', 'django.contrib.auth.views.login',
{'template_name': 'auth/login.html'}, name='login'),
    url(r'^logout/$', 'django.contrib.auth.views.logout',
{'next_page': '/'}, name='logout'),
    url(r'^cuadro_control/', include('cuadro_control.urls')),
    url(r'^monitorizacion/', include('monitorizacion.urls')),
    url(r'^graficos/', include('graficos.urls')),
    url(r'', include('monitorizacion.urls')),
)
```

Las URLs descritas en el fragmento de código anterior, por orden (es importante el orden, ya que el URL *dispatcher* de Django comprueba estos patrones por órden), son las siguientes:

- La primera de ellas, grappelli, hace referencia a una extensión para el admin de Django, que permite personalizarlo y modificar ciertos aspectos del mismo. Esta es la manera en la que se incluye la extensión, justo antes del admin de Django.
- El segundo es el admin de Django, herramienta sobre la que se ha hablado en el capítulo 2 de este documento. Aquí configuramos que se llega a esta funcionalidad a través de /admin.
- El tercero y cuarto son el *login* y el *logout*. Django provee de la funcionalidad de inicio de sesión, además de una plantilla estándar para la introducción de los datos de inicio de sesión. Sin embargo, en este caso se ha utilizado una plantilla de inicio de sesión diferente (ver Ilustración 21), por tanto hay que indicar la ruta ('auth/login.html'). Para realizar el *logout*, al igual que con el *login*, se utiliza la funcionalidad implementada por Django.
- Los 3 patrones siguientes referencian a los módulos urls.py de cada una de las aplicaciones. Con ellos se especifica que /cuadro_control te dirige a todas las urls que haya en esa aplicación, y así sucesivamente.

- El último patrón es el patrón por defecto. Es el comportamiento que se sigue cuando sólo se pone la dirección del sitio web. En este caso se ha configurado para que el sitio por defecto (*index*) sea el de monitorización.

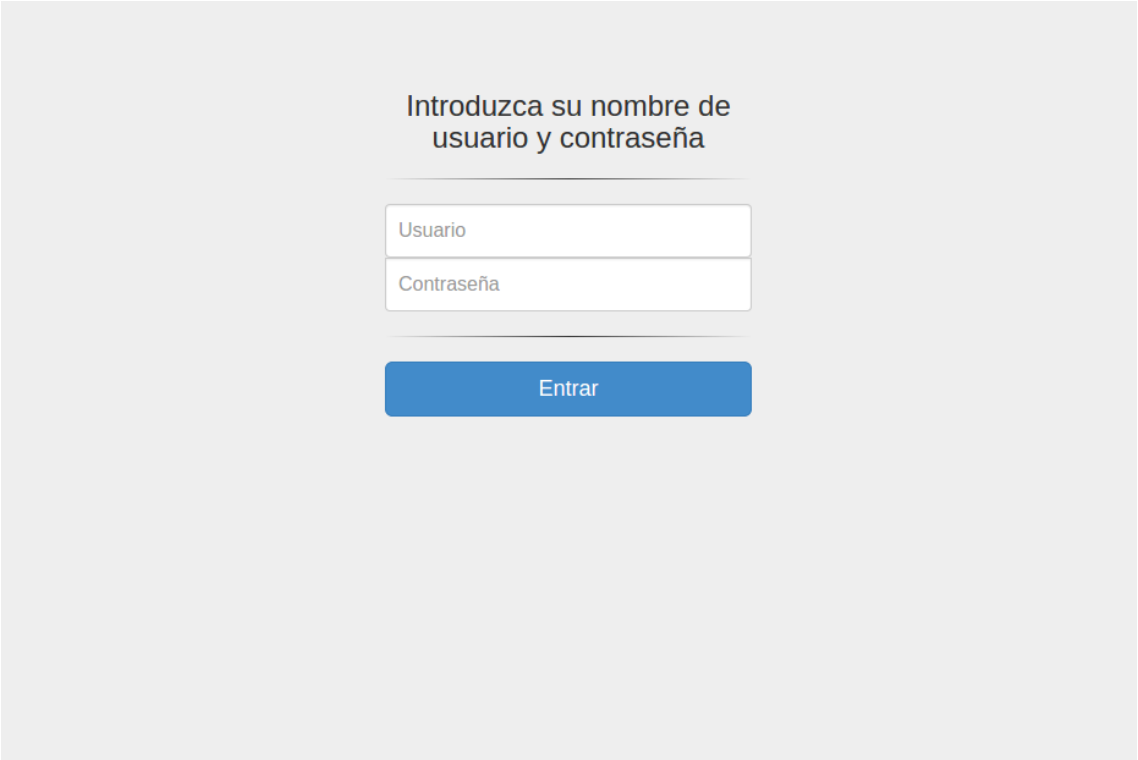

Ilustración 21: Página de login del sitio Smart Wetland.

6.3.1.3 Configuración de los directorios del proyecto

Dado que a la hora de subir un proyecto Django a un servidor web, los ficheros estáticos y los no estáticos van de forma separada, hay que realizar una configuración especial a lo largo del proyecto para que el servidor web pueda encontrar estos ficheros estáticos y sepa a dónde pertenecen.

La primera tarea es organizar los directorios del proyecto. Existe un directorio *static* global para el proyecto en el que están situados los ficheros estáticos comunes para todas las sub-aplicaciones de la página. Después, dentro de cada sub-aplicación, hay un directorio *static* propio para la misma.

A continuación, en el módulo settings.py hay que añadir variables que identifican esto, que son las siguientes:

```
STATICFILES_DIRS = (
    path.join(PROJECT_PATH, 'static'),
)

STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
)
```

La primera variable añade el directorio *static* global del proyecto a la ruta dónde se encuentran los ficheros estáticos. La segunda variable importa los módulos de Django que incluyen algoritmos para buscar directorios llamados *static* dentro de cada una de las sub-aplicaciones del proyecto.

De esta manera, una vez está todo configurado, al escribir el siguiente comando:

manage.py collecstatic

Se añaden todos los archivos estáticos de todo el proyecto a un sólo directorio. En un HTML, cuando se quiera importar algún fichero estático, se hace de la siguiente manera:

```
<script src="{{ STATIC_URL }}js/events.js"></script>
```

6.3.1.4 Configuración del sitio de administración

Para activar el sitio de administración que ofrece el *framework* Django se ha seguido el procedimiento estándar, junto con la configuración propia personalizada para este proyecto en concreto.

La configuración estándar incluye los siguientes pasos:

- Añadir el módulo django.contrib.admin a la variable INSTALLED_APPS antes de hacer *manage.py syncdb*. Esto crea las tablas correspondientes a la administración en la base de datos.
- Añadir las URLs correspondientes al admin en el módulo global del proyecto urls.py.
- Como paso adicional en este caso, se ha incluido la extensión grappelli, que personaliza el aspecto visual de la herramienta de administración de Django. Para incluirlo se realizan los mismos pasos que para el admin de Django.

La configuración adicional es necesaria para incluir la información del resto de modelos del proyecto dentro del admin. Para ello, es necesario crear un archivo admin.py dentro de una sub-aplicación y configurar el contenido. A modo de ejemplo, se muestra a continuación la configuración realizada para la sub-aplicación cuadro_control:

```
class CuadroControlAdmin(admin.ModelAdmin):
      list_filter = ('created_at',)
      search_fields = ('created_at', 'employer__username',
'employer__first_name', 'employer__last_name',)

      fieldsets = [
            (u'Usuario que realizó los cambios', {
                  'classes': ('grp-collapse grp-open',),
                  'fields': ('employer', 'created_at'),
            }),
            (u'Modo de operación', {
                  'classes': ('grp-collapse grp-open',),
                  'fields': ('op_mode',),
            [...]
       ]

admin.site.register(ClientPacket, CuadroControlAdmin)
```

En el código se modifican ciertos aspectos de la vista original, que son las siguientes:

- El campo por el que se desea ordenar los datos. En este caso se elige la fecha de creación.
- También se incluyen los campos por los que se puede hacer la búsqueda.
- A continuación se organizan los datos en grupos y se añade la opción de que se puedan minimizar estos grupos y expandirlos.

A continuación se muestra el resultado de este código en la aplicación:

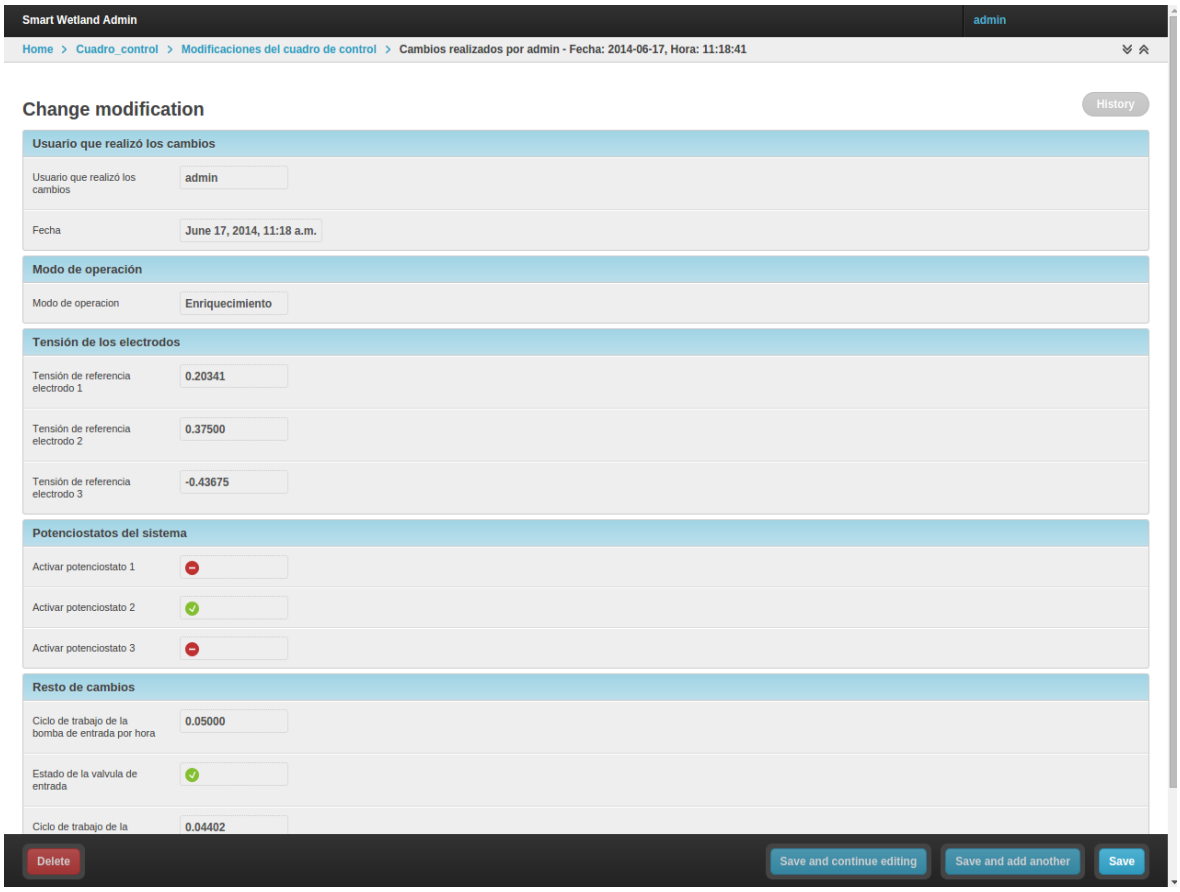

Ilustración 22: Vista del modelo de datos del cuadro de control dentro del sitio de administración.

6.3.2 IMPLEMENTACIÓN DEL MÓDULO MONITORIZACIÓN

Para la implementación de este módulo se han seguido varias etapas, que han sido las siguientes:

- Implementación de la visualización de datos en tiempo real con Node.js y socket.io. Esta implementación ha sido explicada anteriormente.
- Implementación del modelo de datos que siguen los datos provenientes del humedal. Este modelo fue explicado en el capítulo 5 de este documento.
- Implementación de la interfaz gráfica codificando la plantilla HTML y asignando el id de cada campo dentro de la plantilla.

A continuación se muestra el resultado conseguido para la vista de la monitorización.

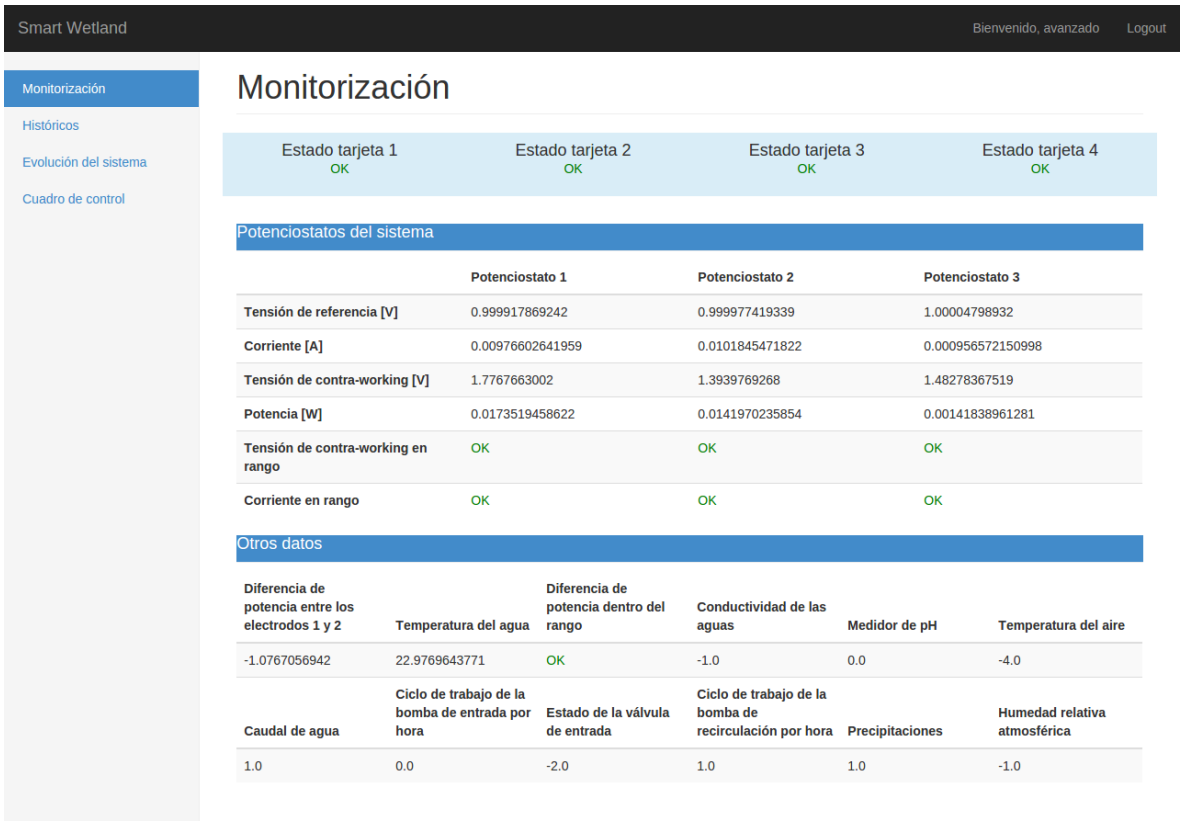

Ilustración 23: Interfaz gráfica de la vista de monitorización.

6.3.3 IMPLEMENTACIÓN DEL MÓDULO GRÁFICOS

Para la implementación de este módulo ha habido 2 tareas principales:

- Desarrollo y codificación de las funciones que implementan la vista: consultas a la base de datos, agrupación de datos según la precisión temporal solicitada, etc.
- Configuración de las librerías Javascript para la representación de gráficos.

A continuación se muestran los pasos principales para la implementación de ambas tareas para cada uno de los gráficos representados en la aplicación.

Uno de los gráficos representados en la aplicación es el que representa la evolución del sistema, implementado con la librería de Javascript Highstock. La función de la vista que obtiene el conjunto de datos a representar de la base de datos es la siguiente:

```
@login_required
def global_plot(request):

      data = RealtimeData.objects.all()

      dataset1 = {'data': sort_data_by_day(data, 'vref_1'),
'name': 'Tensión 1'}
      dataset2 = {'data': sort_data_by_day(data, 'i_1'), 'name':
'Corriente 1'}

      plot_data = [dataset1, dataset2]

      return render(request, 'graficos/representar.html', {
          'user': request.user,
          'plot_data': json.dumps(plot_data,
cls=DjangoJSONEncoder),
      })
```

El primer paso que realiza esta función es la obtención de todos los datos disponibles en la base de datos. A continuación agrupa estos datos por día, para obtener la media del día. Esta funcionalidad se consigue con la función sort_data_by_day, que se muestra a continuación:

```
      for item in data:
            day = datetime.date(item.created_at.year,
item.created_at.month, item.created_at.day)
            timestamp = time.mktime((day).timetuple()) * 1000

            if timestamp in sorted_data:
                  sorted_data[timestamp].append(getattr(item,
data_opt))
            else:
                  sorted_data[timestamp] = [getattr(item,
data_opt)]

      sorted_data = OrderedDict(sorted(sorted_data.items(),
key=lambda t: t[0]))

      [...]
```

Esta es la primera parte de la función, y es la encargada de agrupar los datos por día. Va recogiendo el campo fecha (created_at) de cada dato, para después ir introduciendo cada dato con la clave (el día) correspondiente. A medida que se recorre el set de datos, si algún día ya está en el diccionario que se está creando, simplemente se añade al array que corresponde con esa clave. Por último, se ordena el diccionario por día, ya que Python no ordena los diccionarios de forma automática.

El resto de la función no se muestra en el fragmento anterior. Esta parte recorre el nuevo diccionario creado y crea uno nuevo, realizando la media de cada día. Este es el set de datos que se envía en formato JSON para ser representado.

Para la configuración del gráfico Highstock se han implementado funcionalidades adicionales. A continuación se muestran fragmentos de código significativos:

```
tooltip: {
      formatter: function() {

            var serie = this.points[i].series;
            var point = this.points[i]
            var message = '<span style="color:' + serie.color +
      '">' + serie.name + '</span>: <b>' +
      ((point.point.real_value)).toExponential(3) + ' ' +
      point.point.unit + '</b><br/>'

            return message;
      },
      valueDecimals: 3
}
```

Este fragmento implementa el mensaje que aparece al arrastrar el ratón por encima de un punto dentro del gráfico. Este es uno de los campos de la variable JSON que define el gráfico. El procedimiento es simple: se obtienen el dato que previamente se envió (tal y como se mostró en fragmentos de código anteriores). En este dato está toda la información que se va a mostrar en el *tooltip*, por tanto lo único que hay que hacer es recoger los campos *unit*, que definen las unidades de la variable en cuestión, el valor, *real_value*.

También se han implementado varias configuraciones de Highstock, tal y como es el nombre de los ejes X e Y, la utilización de 2 ejes Y, uno para el voltaje y otro para la corriente y otras configuraciones estándar de la librería. Se adjunta una captura del gráfico obtenido:

Ilustración 24: Representación del gráfico Evolución del sistema.

6.3.4 IMPLEMENTACIÓN DEL MÓDULO CUADRO DE CONTROL

Las etapas que existen para la implementación de este módulo son las siguientes:

- Desarrollo de la interfaz gráfica, formulario HTML y funciones Javascript adicionales.
- Traducción de los datos provenientes del formulario para la creación de una trama hexadecimal y su posterior envío al sistema de control de la WSN a través de la red.

La primera de las etapas consiste en la codificación del formulario HTML que se muestra al usuario, además de indicarle los últimos cambios enviados al servidor. Para crear el formulario HTML se ha escogido el tipo de *widget* de entrada de datos para cada tipo de datos. Los escogidos en este caso han sido los siguientes:

- Para los campos que indican valores numéricos decimales se ofrece la posibilidad de introducirlos con un *slider* o de forma manual.

- Para los campos de tipo *booleano*, como por ejemplo, los que indican activado o desactivado, se han implementado con un campo tipo *select*.

Para la implementación del *slider* mencionado se ha escrito una función sencilla en Javascript que se encarga de actualizar el valor numérico de forma visual según se mueve la barra. La función desarrollada es la siguiente:

```
function outputUpdate(val, selector) {
      document.querySelector('#id_vref_' + selector).value = val;
};
```

Dado que hay 3 campos llamados id_vref_X en los que se implementa el *slider*, al codificar en campo HTML de tipo *range*, se añade un parámetro denominado onchange que llama a esta función, indicando el selector correspondiente al campo.

La manera en la que se implementa la visualización de los últimos cambios es mediante código Javascript. En la vista de Django que se ejecuta antes de enviar el HTML procesado al cliente, se hace una consulta a la base de datos que obtiene los últimos datos en la tabla de cambios en el cuadro de control (esta función también envía una variable que indica respecto al estado del servidor al que se van a enviar los cambios). Este código asigna el valor inicial de cada campo al correspondiente. También se añade este valor al lado de los títulos, para mayor claridad visual. A continuación se muestra la interfaz visual conseguida con este proceso.

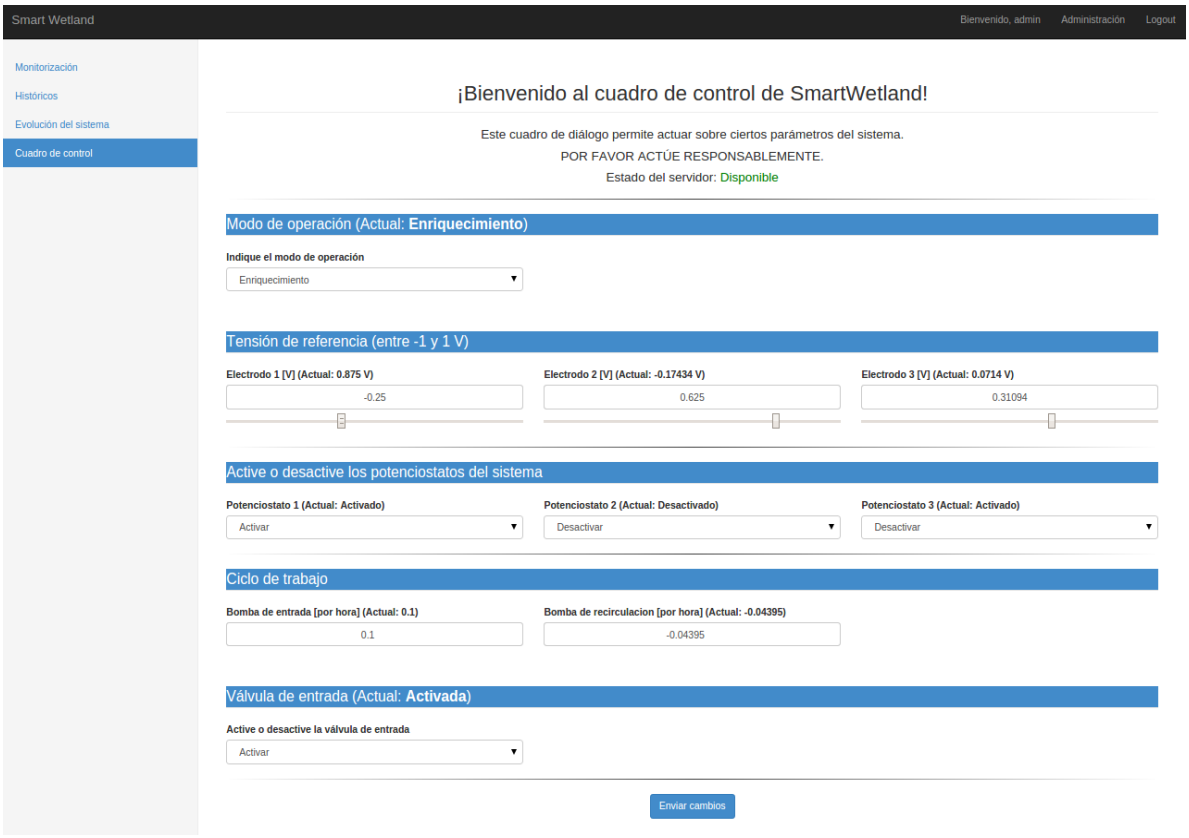

Ilustración 25: Interfaz gráfica de la vista del cuadro de control.

Para el envío de los datos hexadecimales al servidor a través de la red se implementa una función en la vista de esta sub-aplicación. Esta función es la misma que la que se ejecuta para obtener los últimos cambios realizados en el servidor, información de la conexión, obtención del formulario, etc.

Esta función ejecuta un código adicional si el método con el que se obtiene la página es POST (normalmente para sólo obtener la página la petición se hace con GET). También se comprueba si el formulario es correcto, al igual que lo que se hizo en la sub-aplicación monitorización para validar los datos.

Finalmente, se abre un socket TCP en la dirección y puerto en el que está situado el servidor de la WSN (aún no definidos) y se realiza la conversión de cada dato a hexadecimal con la función de Python struct.pack (el proceso es similar a lo que se explicó anteriormente para el servidor intermediario en la conversión de datos hexadecimales provenientes de la WSN).

6.4 DESPLIEGUE DE LA APLICACIÓN EN EL SERVIDOR WEB

Tal y como se explicó en el capítulo 4, la configuración escogida para actuar como servidor web es la de Gunicorn junto con Nginx como servidor *proxy.*

La configuración establecida en el servidor principal (Nginx) es la siguiente:

```
# smartwetland

server {
    # Puerto en el que escucha el servidor Nginx
    listen      8000;
    # Nombre del dominio en el que sirve Nginx
    server_name smartwetland.com;
    charset     utf-8;
    # Tamaño máximo de subida
    client_max_body_size 75M

    location /static/ {
        alias /home/rodrigo/Proyectos/smart_wetland/static-dir/;
    }

    # Redirección de las peticiones de ficheros no estáticos
    location / {
        proxy_pass http://127.0.0.1:8001;
        proxy_set_header X-Forwarded-Host $server_name;
        proxy_set_header X-Real-IP $remote_addr;
        add_header P3P 'CP="ALL DSP COR PSAa PSDa OUR NOR ONL UNI
COM NAV"';
    }
}
```

Este es el fichero situado en /etc/nginx/sites-available. Realiza configuraciones básicas del servidor de Nginx, además de la redirección de las peticiones de ficheros no estáticos al puerto 8001, que es dónde está situado Gunicorn.

Para lanzar el proceso de Gunicorn se escribe el siguiente comando en una consola de comandos:

```
gunicorn smart_wetland.wsgi:application -b:8001
```

El módulo wsgi del proyecto Django es utilizado para comunicar a los servidores web que soporten esta tecnología con el código de la aplicación. Con la opción –b se especifica el puerto en el que el servidor web empieza a ejecutarse.

Capítulo 7

Pruebas

En este capítulo se hablará sobre las pruebas realizadas en el sistema para comprobar su funcionalidad.

Las pruebas realizadas se han focalizado en varios puntos en los que se dividirá el capítulo. Primero se explicará cómo se ha probado el sistema de mensajes situado en el *back-end* de la web. Después, se mostrará cómo se ha probado la funcionalidad de aplicación. Por último, se hablará sobre las pruebas realizadas para el testeo de la seguridad, autenticación y permisos de la web.

7.1 PRUEBAS REALIZADAS EN LA COMUNICACIÓN CON LA WSN

Para probar que ambas partes entienden y han implementado el protocolo de comunicación diseñado de forma correcta, se han realizado diversas pruebas y se ha llevado a cabo un proceso de depuración del código desarrollado.

La primera de las pruebas ha sido la de probar la comunicación de ambos programas creando una red local mediante un cable Ethernet cruzado. El nodo central de la WSN se conectaba a un prototipo del humedal real creado para la fase anterior al despliegue del sistema en el humedal real. Las conclusiones que fueron alcanzadas con esta prueba fueron las siguientes:

- El primero de los errores se dio en la forma en la que estaba implementado el protocolo en el prototipo de nodo central del sistema de control. Los

datos no eran transformados previamente a un formato común, sino que eran enviados a través de la red sin transformaciones previas (el protocolo especifica que el formato de los datos deben ser cadenas hexadecimales codificadas en ASCII).

- Otro problema era la implementación de los sockets TCP en C. Debido a la diferencia entre los lenguajes de la red y de la máquina, los caracteres eran desordenados y el mensaje que llegaba al servidor *Broker* era diferente.

Para la solución de este problema, se ha optado por serializar los datos, es decir, cada máquina de forma independiente transforma los datos al formato hexadecimal codificado en ASCII y lo trata como una cadena de texto. De esta manera, no hay ningún problema de compatibilidad entre los mensajes enviados, ya que todos son cadenas de texto, que posteriormente son traducidas.

La segunda prueba realizada en el servidor *Broker* tiene como objetivo garantizar que el servidor recibe los mensajes de manera completa y ordenada. Debido a que en la implementación real pueden existir problemas de latencia y congestión de tráfico, se ha depurado el código de este servidor para que realice esta funcionalidad y recoja los mensajes de forma correcta.

A continuación se muestra el código del script codificado para probar esta funcionalidad:

```
# Primera prueba: se envía de forma separada el int que informa
# sobre el tamaño de la trama y después el mensaje.
network.send_packet('00000110', sample_socket)
network.send_packet(message1, sample_socket)

# Segunda prueba: envío del tamaño para esperar 10 segundos
# y enviar el mensaje.
network.send_packet('00000110', sample_socket)
time.sleep(10)
network.send_packet(message2, sample_socket)

# Tercera prueba: envío de varios mensajes seguidos.
network.send_packet(message1, sample_socket)
network.send_packet(message2, sample_socket)
network.send_packet(message3, sample_socket)
```

Además, como prueba final de esta funcionalidad, se han realizado pruebas con el software Wireshark, herramienta que permite analizar paquetes en la red. En la Ilustración 26 se muestra la captura realizada con esta herramienta para verificar la funcionalidad de la sub-aplicación cuadro de control, en la que cuando el usuario rellena el formulario de la página, se crea un mensaje hexadecimal con destino el nodo central de la WSN. En este caso, dado que es un entorno de pruebas aislado, los mensajes se envían con destino *localhost* al puerto 50007, dónde hay un socket TCP abierto por el servidor *Broker*.

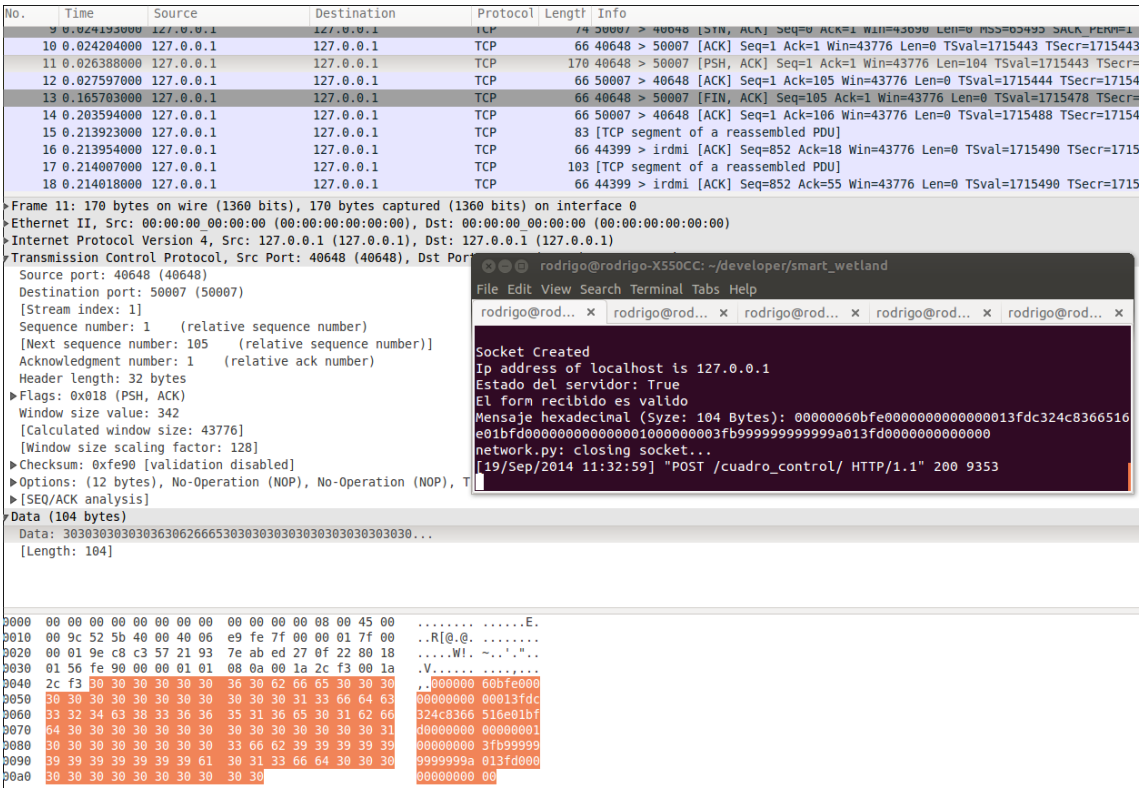

Ilustración 26: Prueba de funcionalidad con Wireshark.

En la captura se puede comprobar que el mensaje es enviado correctamente a través de la red. También se puede comprobar que el mensaje hexadecimal enviado es el mismo al que la aplicación ha creado.

En la Ilustración 27 se muestra otra prueba desarrollada para comprobar que el mensaje hexadecimal es construido correctamente.

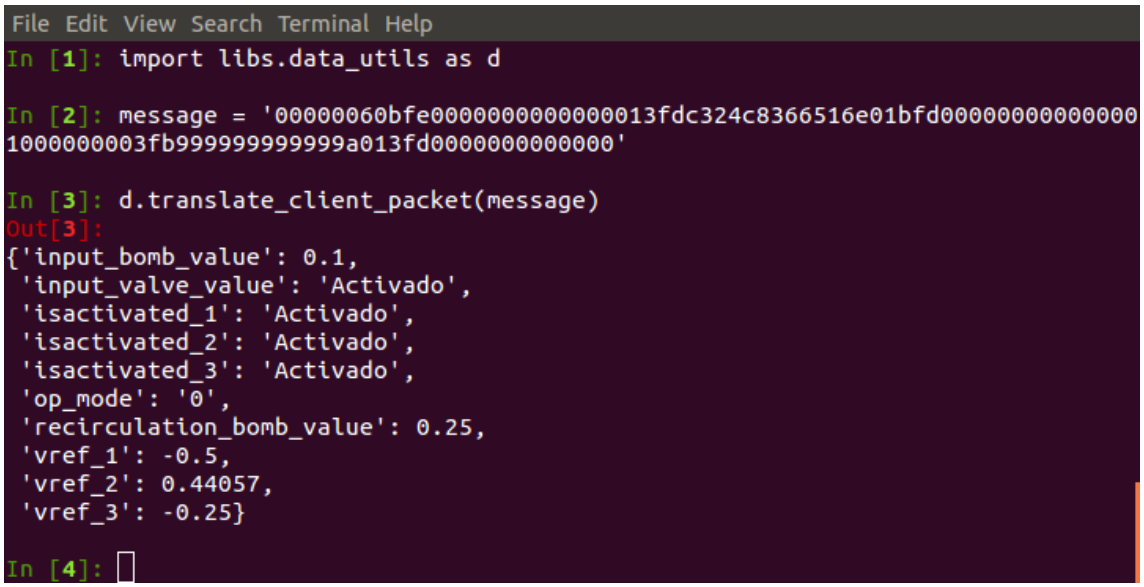

Ilustración 27: Traducción del mensaje hexadecimal creado por el cuadro de control.

En esta imagen se puede ver que el mismo mensaje enviado anteriormente contiene los campos seleccionados previamente en el formulario del cuadro de control.

7.2 PRUEBAS REALIZADAS EN LA APLICACIÓN WEB

En este apartado se explicaran diferentes pruebas realizadas en la aplicación web, destinadas principalmente a comprobar la funcionalidad y la seguridad de la misma.

7.2.1 PRUEBAS DE FUNCIONALIDAD

La funcionalidad de las sub-aplicaciones monitorización y cuadro de control queda probada parcialmente con las pruebas de comunicación con la WSN. Por tanto, queda probar si los datos se muestran de manera correcta y la validación de los datos que el usuario envía a través del cuadro de control.

Para las pruebas de visualización de datos se ha comprobado el comportamiento de la página de monitorización con todo tipo de datos. Dado que el servidor *Broker* comprueba tanto el tamaño como cada uno de los campos del mensaje hexadecimal recibido, el contenido que deberá mostrar en la página está previamente validado.

Para la validación de los datos enviados en el cuadro de control se ha utilizado la herramienta que ofrece Django para tal tarea. Mediante el módulo *forms.py*, se especifica el número de campos que debe venir en la petición POST, el tipo de los mismos y otras restricciones como puede ser el rango de valores admitido.

7.2.1 PRUEBAS DE SEGURIDAD

Las principales pruebas de seguridad realizadas en la aplicación tienen que ver con la autenticación. Cabe mencionar que la validación de datos y URLs en toda la aplicación, además de ser pruebas de funcionalidad, también lo son de seguridad, ya que este tipo de fallos suelen ser vulnerabilidades para ataques de seguridad como los que se ha explicado en el capítulo 2 de este documento. A continuación se enumeran las pruebas realizadas:

- Comprobación de que cada uno de los usuarios no tiene visibilidad de las zonas de la web a las que no tiene permisos para acceder, ni que tampoco pueda acceder introduciendo la URL del sitio de forma manual.
- También se comprueba que, en el caso del cuadro de control, un usuario que construya una petición POST de forma manual y la envíe a la URL correspondiente, no sea capaz de realizar cambios en el sistema.
- Comprobación de que ninguna de las vistas de la página web, excepto el *login*, sea visible sin estar autenticado.

Todas estas pruebas de seguridad han tenido resultados positivos y la seguridad requerida en la página es la que se definió en la etapa de requisitos.

Chapter 8

Conclusions and future works

The content of this chapter will focus on the reached conclusions during the development of the project. In addition, possible future works and additional features related with the developed task will be discussed.

8.1 CONCLUSIONS

This completed final project had some purposes associated with the optimization and the viability of the implementation of constructed smart wetlands in small populations. The developed tasks during this project have been:

- Study of the proposal and the objectives to be achieved in order to make a proper selection of the optimal web development technologies.
- Implementation of the pre-designed protocol for the communication between the Wireless Sensor Network designed and implemented by one of the Smart Wetland project partners, and the remote control system, in addition to the contribution of possible upgrades for this protocol in the future.
- Design and implementation of a TCP/IP proxy server, using Python, responsible for intercommunicating the web application and the Wireless Sensor Network.
- Design and implementation with Python/Django of all modules in the web application (including database management) in charge to receive, display and send data, besides to plot the specified charts with this data.
- Design and implementation of a Node.js server/client in charge to provide the real time data visualization functionality to the web application.

- Deployment and settings of the web application in a Virtual Private Server (VPS) platform with the Nginx and Gunicorn Web Servers.

On the following list, some of the achieved conclusions associated to the project initial purposes are discussed:

- The main conclusion is that a reliable and useful tool for remote controlling has been obtained. In this case, the remote controlled system is a wetland, but any other system with Internet connectivity and the capability to send, receive and process messages could be also remote controlled.
- In addition, the integration of the selected technologies has been successfully proved. The pre-analysed behaviour to the implementation has matched with the real performance of the system.
- Another of the obtained conclusions is that the system functionality will be easily modified or upgraded, if necessary.
- However, the achievement of the initial purposes (related to efficiency and cost savings) directly relies on the performance of the wetland control system. This means that the successful implementation of this remote control system doesn't secure the whole system works. This last point has to be evaluated in upcoming months in the Smart Wetland project.

Other unrelated conclusions to the project initial purposes are:

- At the beginning of the project, all of the used technologies were completely unknown. At the ending of it, extended knowledge about some of the most powerful technologies and web development tools has been acquired. Besides, some of previously studied networking concepts have been mastered.
- This completed project has been profitable for the student. It has provided a lot of self-learning and information search and synthesis (from different sources) capabilities. Other profitable things of the project have been: technical knowledge, constant work and professional skills, larger interest for researching and new professional objectives.

8.2 FUTURE WORKS

Some of possible future works related to the extended functionality of the system are:

- Implementation of a settings menu, allowing the user to set some self-control rules. For example, if some of the quality parameters goes down to certain level, the system should automatically send certain changes to the control system.
- Include other kind of charts, depending on the future project needs.

- Implementation of an alarm system. It should send warnings whenever a certain variable exceeds a defined threshold, for example.
- Allow the control of more than one wetland from the web page. This could centralize the control of a huge geographically dispersed amount of wetlands.

Other proposed future works are:

- Migration of the developed system to a mobile application. This kind of application could be very useful in the future, given the workers tendency of having this kind of tools in their mobile phones or tablets.
- Implementation of this tool in other ambient monitoring environments. The WSN technology is rising and related projects are emerging, so this tool could be a good choice for an enterprise to merge all its projects to one remote control application.

Planificación y presupuesto

PLANIFICACIÓN

A continuación se describe la planificación y descripción detallada de todas las tareas realizadas para la elaboración de este TFG.

Actividad 1. Análisis y estudio de las tecnologías a utilizar.

- **Actividad 1.1. Análisis de las tecnologías a utilizar:** En esta actividad se estudia el estado del arte en el desarrollo web. A partir del análisis de requisitos planteado, se estudian las tecnologías óptimas para la implementación del sistema de telecontrol.
  Duración de la actividad: 2 semanas.
- **Actividad 1.2. Aprendizaje y estudio de las tecnologías seleccionadas:** Durante el desarrollo de esta actividad, se ha llevado a cabo un proceso de aprendizaje y recopilación de información para el uso de las tecnologías elegidas en la actividad 1.1, tales como el lenguaje de programación Python, Django, Node.js y otro tipo de conocimientos necesarios para el desarrollo del TFG.
  Duración de la actividad: 4 semanas.

Actividad 2. Diseño de la aplicación web y de la arquitectura del sistema.

- Actividad 2.1. Diseño de los modelos de datos a utilizar.
  Duración de la actividad: 1 semana.
- Actividad 2.2. Diseño de la estructura del sistema de telecontrol.
  Duración de la actividad: 2 semanas.
- Actividad 2.3. Diseño de la arquitectura del proyecto Django.

Duración de la actividad: 1 semana.

Actividad 3. Implementación de la aplicación web.

- Actividad 3.1. Implementación del protocolo de comunicación diseñado para la comunicación con la WSN.
  Duración de la actividad: 2 semanas.
- Actividad 3.2. Implementación del servidor TCP/IP intermediario (*Broker).*
  Duración de la actividad: 2 semanas.
- Actividad 3.3. Implementación de la sub-aplicación del cuadro de control.
  Duración de la actividad: 3 semanas.
- Actividad 3.4. Implementación de la sub-aplicación de monitorización.
  Duración de la actividad: 1 semana.
- Actividad 3.5. Implementación del servidor y cliente Node.js.
  Duración de la actividad: 3 semanas.
- Actividad 3.6. Implementación de la sub-aplicación de representación de gráficos.
  Duración de la actividad: 2 semanas.

Actividad 4. Realización de pruebas y despliegue de la aplicación.

- Actividad 4.1. Realización de pruebas con el primer prototipo de nodo central de la WSN.
  Duración de la actividad: 2 semanas.
- Actividad 4.2. Pruebas de seguridad y funcionalidad de la aplicación.
  Duración de la actividad: 1 semana.
- Actividad 4.3. Configuración y despliegue de la aplicación con Nginx y Gunicorn.
  Duración de la actividad: 1 semana.

Actividad	Duración
Actividad 1	
Actividad 1.1	2 semanas
Actividad 1.2	3 semanas
Actividad 2	
Actividad 2.1	1 semana
Actividad 2.2	2 semanas
Actividad 2.3	1 semana
Actividad 3	
Actividad 3.1	2 semanas
Actividad 3.2	2 semanas

Actividad 3.3	2 semanas
Actividad 3.4	1 semana
Actividad 3.5	2 semanas
Actividad 3.6	2 semanas
Actividad 4	
Actividad 4.1	2 semanas
Actividad 4.2	1 semana
Actividad 4.3	1 semana
Total	24 semanas

El desarrollo de las tareas ha tenido lugar durante los meses de Abril a Agosto de 2014. Por tanto, el desarrollo completo de la tarea se ha completado en 5 meses. A continuación se muestra el diagrama de Gantt correspondiente a la división de tareas (el eje X designa el número de la semana, correspondiendo la semana 0 a la primera semana de Abril de 2014 y la semana 24 a la última semana de Agosto de 2014):

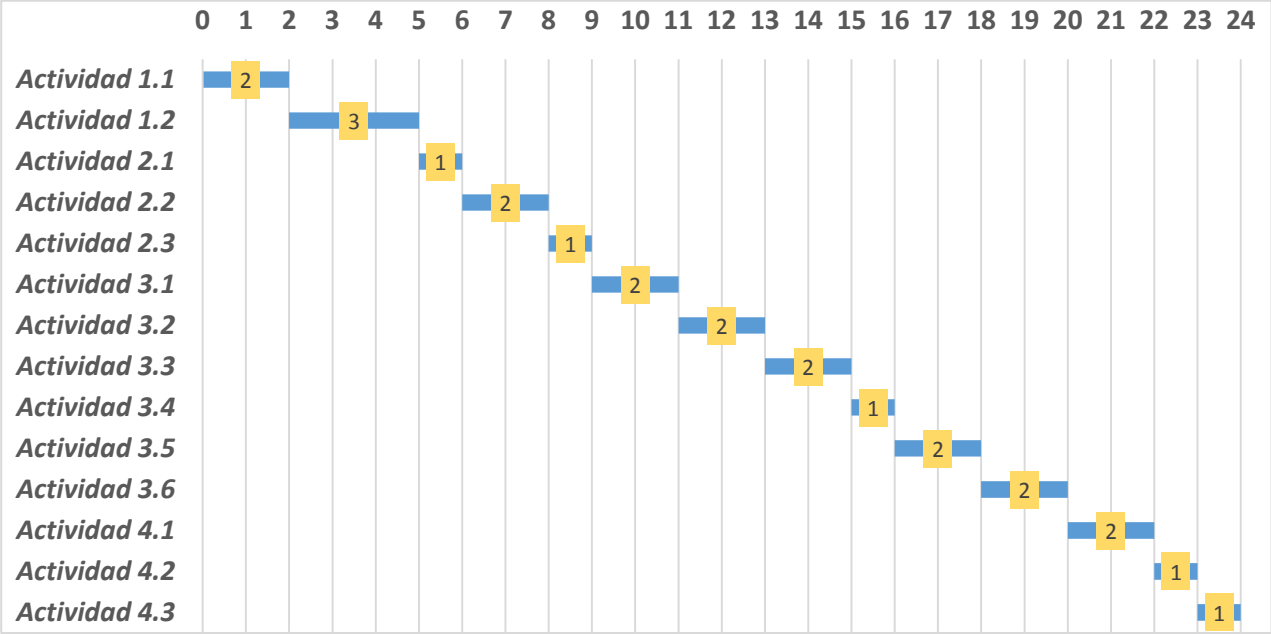

Tabla 3: Diagrama de Gantt del proyecto.

PRESUPUESTO

En esta sección se calculará el presupuesto necesario para la realización de la tarea descrita a lo largo de este documento.

Como primer paso, se hace un cálculo de las horas productivas en un año por un trabajador en una empresa.

Horas productivas en un año	
Días totales en un año	365 días
Fines de semana	104 días
Vacaciones	22 días
Días festivos	12 días
Enfermedad/Otros	2 días
Días productivos en un año	**225 días**
Total en horas (8h/día)	**1800 horas**

Tabla 4: Estimación de horas productivas de un trabajador en un año.

Dado que la duración del proyecto han sido 5 meses, se obtiene que el total de horas dedicadas al mismo asciende a las 750 horas. Con el dato del coste de un trabajador en la empresa por hora (35€/hora), se obtiene la siguiente tabla relativa a los costes directos de personal:

Nombre del trabajador	Puesto	Dedicación (nº de horas)	Coste (1 hora)	Coste total (Euros)
Rodrigo Argüello Flores	Ingeniero Telemático	750	35	26250

Tabla 5: Costes directos asociados al personal.

A continuación se describen, de forma detallada, los costes de material asociados a la tarea realizada (los costes con 5 meses de duración están asociados únicamente a la fase descrita en esta memoria; los que tienen 12 meses de duración están asociados a

la fase de despliegue de la aplicación dentro del sistema global del proyecto *Smart Wetland*):

Descripción	Coste (Euros)	%Uso dedicado al proyecto	Dedicación (meses)	Período de depreciación (meses)	Coste imputable (Euros)
Ordenador sobremesa	400	40	5	60	50,66
Ordenador portátil	1000	70	5	60	641,66
Ordenador industrial	1200	100	12	60	960
Total					1652,32

Tabla 6: Costes directos asociados al material.

Además, habría que añadir otros costes, que se describen a continuación:

Descripción	Coste (Euros)	%Uso dedicado al proyecto	Dedicación (meses)	Período de depreciación (meses)	Coste imputable (Euros)
Virtual Private Server	10	-	12	-	120
Total					120

Tabla 7: Otros costes directos.

Los costes indirectos (facturas de luz, conexión a internet, transporte, etc.) se traducen en un 20% del total de los otros costes directos. Por tanto, el presupuesto total para la tarea descrita es el siguiente:

Costes	Costes totales (Euros)
Costes directos de personal	26250
Costes directos de material	1652,32
Otros costes directos	120
Costes indirectos (20%)	5604,46
Total proyecto	**33626,78**

Tabla 8: Presupuesto total.

Marco regulador y entorno socioeconómico

MARCO REGULADOR

El marco regulador en el que se encuentra este TFG está relacionado con el correcto el tratamiento de las aguas residuales urbanas y la posterior calidad de las aguas vertidas por el humedal a las cuencas fluviales próximas.

Dado que el sistema de telecontrol desarrollado tendrá la capacidad de modificar parámetros de depuración del humedal, un mal diseño en la seguridad y mantenimiento del mismo podría conllevar a modificaciones del proceso de depuración de las aguas por personal no autorizado y conllevar al vertido de aguas contaminadas o que no cumplen los estándares de calidad del agua a los ríos próximos al mismo.

Por tanto, entran en juego leyes relacionadas con la calidad de la depuración del agua y el control de vertidos tales como:

- Real Decreto 1290/2012, de 7 de septiembre, por el que se modifica el Reglamento del Dominio Público Hidráulico, aprobado por el Real Decreto 849/1986, de 11 de abril, y el Real Decreto 509/1996, de 15 de marzo, de desarrollo del Real Decreto-ley 11/1995, de 28 de diciembre, por el que se establecen las normas aplicables al tratamiento de las aguas residuales urbanas. [17]

También se ha de tener en cuenta la legislación vigente de la zona, dependiendo de la cuenca hidrográfica donde se produzcan los vertidos del humedal. Se expone, a modo

de ejemplo, el plan hidrológico vigente en el río Guadalquivir, ya que es la cuenca fluvial donde se producen los vertidos del humedal prototipo del proyecto *Smart Wetland:*

- Real Decreto 355/2013, de 17 de mayo, por el que se aprueba el Plan Hidrológico de la Demarcación Hidrográfica del Guadalquivir. [18]

ENTORNO SOCIOECONÓMICO

El entorno socioeconómico asociado al TFG está situado de las pequeñas poblaciones dentro de España. Gran cantidad de estos territorios sufren un mal tratamiento de sus aguas residuales.

Se estima entre 3 y 4 millones de habitantes equivalentes la carga equivalente que todavía no recibe un tratamiento adecuado en poblaciones por debajo de los 2.000 habitantes equivalentes. Aunque esta carga representa un pequeño porcentaje del total de la carga contaminante a nivel nacional, el número de aglomeraciones afectadas se estima en más de 6.000, la mayoría de ellas menores de 500 habitantes equivalentes. [20]

Esto ocurre debido a que estas poblaciones no disponen de recursos económicos suficientes para asumir los costes asociados a la implantación y mantenimiento de las plantas de depuración convencionales.

Por tanto, esto se traduce en que estas poblaciones están realizando vertidos en cuencas fluviales que no cumplen los estándares de calidad del agua establecidos en España.

Referencias

[1] Alonso, J. A. (16 de Mayo de 2013). *Humedales artificiales como sistemas naturales de depuración de aguas residuales. Conceptos e historia.* Obtenido de http://www.madrimasd.org/blogs/remtavares/2013/05/16/131891

[2] Anders, M. (2011). *Python 3 Web Development Beginner's Guide.* Packt Publishing.

[3] Busalmen, J. P.-N. (2010). Approach to Microbial Fuel Cells and Their Applications. *Fuel Cell Science*.

[4] Chafloque, W. A., & Gómez, E. G. (2006). Diseño de humedales artificiales para el tratamiento de aguas residuales en la UNMSM. *Revista del Instituto de Investigación de la Facultad de Ingeniería Geológica, Minera, Metalurgica y Geográfica*.

[5] Close, T. (Octubre de 2014). *The Confused Deputy rides again!* Obtenido de http://waterken.sourceforge.net/clickjacking/

[6] Crockford, D. (2008). *JavaScript: The Good Parts.* Yahoo! Press.

[7] *Django documentation*. (s.f.). Obtenido de https://docs.djangoproject.com/

[8] *Django Project*. (s.f.). Obtenido de https://www.djangoproject.com/

[9] Fjordvald, M. (2013). *Instant Nginx Starter.* Packt Publishing.

[10] Greenfeld, D., & Audrey, R. (2014). *Two Scoops of Django: Best Practices for Django 1.6.* Two Scoops Press.

[11] *Gunicorn documentation*. (s.f.). Obtenido de http://docs.gunicorn.org/en/latest/index.html

[12] Kaplan-Moss, A. H. (2009). *The Django Book.* Apress.

[13] Kiessling, M. (2014). *The Node Beginner Book.* Leanpub.

[14] *Licencia Pública General de GNU*. (s.f.). Obtenido de http://www.gnu.org/licenses/licenses.es.html#GPL

[15] Llagas Chafloque, W. A., & Guadalupe Gómez, E. (2006). *Revista del Instituto de Investigaciones FIGMMG.* Obtenido de http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/geologia/vol9_n17/a11.pdf

[16] Miguel, C. (30 de Enero de 2013). *iagua: Los humedales artificiales*. Obtenido de Los humedales artificiales: http://www.iagua.es/blogs/carolina-miguel/los-humedales-artificiales-componentes-y-tipos

[17] Ministerio de agricultura, a. y. (2012). *Real Decreto 1290/2012 de 7 de septiembre.*

[18] Ministerio de agricultura, a. y. (2013). *Plan Hidrológico aprobado por R.D. 355/2013 de la demarcación del Guadalquivir.*

[19] Ministerio de agricultura, alimentación y medio ambiente. (2013). *Gestión de aguas residuales en pequeños núcleos urbanos.*

[20] Ministerio de Medio Ambiente. (2007-2015). *Plan Nacional de Calidad de las Aguas: Saneamiento y Depuración.*

[21] Mohanakrishna, G. a. (2013). "Multiple process integrations for broad perspective analysis of fermentative H2 production from wastewater treatment: Technical and environmental considerations. *Applied Energy*.

[22] *Nginx documentation*. (s.f.). Obtenido de http://nginx.org/en/docs/

[23] *Python Software Foundation License Version 2*. (s.f.). Obtenido de http://opensource.org/licenses/Python-2.0

[24] *Repositorio oficial de Django Debug Toolbar*. (s.f.). Obtenido de https://github.com/django-debug-toolbar/django-debug-toolbar],

[25] Romero-Aguilar, M., Colín-Cruz, A., Sánchez-Salinas, E., & Ortiz-Hernández, M. L. (2009). Tratamiento de aguas residuales por un sistema piloto de humedales artificiales: evaluación de la remoción de la carga orgánica. *Revista internacional de contaminación ambiental*.

[26] *Sitio web oficial del grupo Bioelectrogénesis de IMDEA-Agua*. (s.f.). Obtenido de www.bioelectrogenesis.com

[27] *Sitio web oficial del grupo de investigación de MFC*. (s.f.). Obtenido de http://www.microbialfuelcell.org

[28] Sriparasa, S. S. (2013). *JavaScript and JSON Essentials.* Packt Publishing.

[29] *V8 JavaScript Engine Project Home.* (s.f.). https://code.google.com/p/v8/.